Trabajo de Fin de Grado

Grado en Ingeniería Informática
Computación

Aplicativo de análisis automático de entrenamientos

23 de octubre de 2020

Autor: David Mingueza Álvarez de la Campa
Director: Ricard Pérez Pinazo
Ponente: Enrique Romero Merino

Resumen

Trainerer es una empresa que crea planes de entrenamiento para deportistas. Para ello disponen de una plataforma que importa datos de aplicaciones que registran entrenamientos como Strava y Garmin. Con estos datos se analizan los registros de las actividades y se guardan para crear un historial deportivo de cada atleta.

Con el desarrollo de este proyecto se pretende dar solución al problema de analizar automáticamente el estado de forma de un deportista y predecir la evolución de sus parámetros de rendimiento a corto plazo.

Para llevar a cabo esta tarea se ha utilizado el lenguaje de programación Python, juntamente con el framework Flask y el sistema de contenedores Docker.

Resum

Trainerer és una empresa que crea plans d'entrenament per esportistes. Per això disposen d'una plataforma que importa dades d'aplicacions que registren entrenaments com Strava y Garmin. Amb aquestes dades s'analitzen els registres de les activitats i es guarden per crear un historial esportiu de cada atleta.

Amb el desenvolupament d'aquest projecte es prenten proporcionar una solució al problema d'analitzar automàticament l'estat de forma d'un esportista i predir l'evolució dels seus paràmetres de rendiment en un període de temps curt.

Per dur a terme aquesta tasca s'ha utilitzat el llenguatge de programació Python, juntament amb el framework Flask i el sistema de contenidors Docker.

Abstract

Trainerer is a company that makes training plans for athletes. To this end, they have a platform that imports data from external applications that register training like Strava and Garmin. The logs of the activities are analyzed with this data and they are saved to create a sporty track record for each athlete.

With the development of this project, it is intended to provide a solution to the problem of analysing automatically the condition of an athlete and predict the evolution of their performance parameters in a short term.

To perform this task, the Python programming language has been used mainly, besides the Flask framework and the Docker container system.

Índice

1. Introducción

El mundo del deporte está experimentando grandes cambios a nivel tecnológico. Desde la introducción del VAR en el fútbol hasta pequeños aparatos que miden una gran cantidad de constantes físicas durante un entrenamiento.

No obstante, un gran problema que todavía no se ha solucionado es conocer con exactitud el rendimiento obtenido de cada entrenamiento que se realiza y la mejora que está aportando.

Es por eso que la empresa Trainerer [1] inevitablemente se encontró con este problema. Gran parte de sus clientes siempre llegaban a formular las mismas preguntas como por ejemplo: *¿Cómo estoy?, ¿Cómo me ves?, ¿Cómo lo he ido haciendo durante estos días?* Las cuales son bastante difíciles de responder viendo las grandes cantidades de datos que están relacionadas.

Es cierto que existen aplicaciones que ayudan a visualizar los datos que se recogen en forma de gráficas y estadísticas [2]. De ahí se puede extraer mucha información como por ejemplo si el deportista ha cumplido con la planificación del entreno, cuál ha sido su mejor sector y que parte ha sido más costosa, entre otros. No obstante, no siempre las gráficas ni las constantes ayudan a contestar estas preguntas con exactitud. A veces, se pueden encontrar problemas como pérdida de la señal GPS del dispositivo durante la sesión que genera pérdida de información que puede ser importante, valores anormales en las pulsaciones del deportista debido a un desplazamiento involuntario del sensor y más. Estos factores hacen más compleja esta tarea, la cual para poder realizarse de forma óptima necesita tiempo y paciencia.

Este proyecto da una respuesta plausible a este problema proporcionando una solución que muestra a un deportista su estado de forma y la evolución de su rendimiento hacia sus objetivos.

Para ello el proyecto deberá ser escalable, rápido y eficaz en todas sus funcionalidades y ser capaz de manejar grandes cantidad de datos.

2. Contexto

2.1 Actores implicados

En este proyecto habrá agentes que estarán directamente implicados en su desarrollo, además de los agentes que se beneficiarán y utilizaran los resultados:

- **Trainerer** - Es la empresa donde se realizará el proyecto. Las nuevas funcionalidades se implementarán dentro de su aplicación web así que la empresa saldrá totalmente beneficiada con los resultados.

- **Investigador** - El principal responsable de planificar, desarrollar y documentar el proyecto, así como procesar los resultados y sacar conclusiones.

- **Director** - El responsable de supervisar al investigador. Planificará las tareas por adelantado para poder aconsejar, dirigir y ayudar en el desarrollo del proyecto. Además es el jefe ejecutivo de la empresa por lo que implícitamente saldrá muy beneficiado con los resultados.

- **Ponente -** Profesor de la Facultad de Informática de Barcelona que supervisará el proyecto para asegurarse de que en todo momento se cumplen los requisitos necesarios de un Trabajo de Fin de Grado. Será el encargado de aconsejar al investigador sobre la parte científica y guiarle en su desarrollo.

2.2 Justificación

Actualmente, no existe ningún software que implemente o sea similar a la funcionalidad que se quiere desarrollar. Haciendo una búsqueda rápida se pueden encontrar programas y dispositivos que tienen aplicaciones donde se analiza el rendimiento en un entrenamiento a través de valores y gráficas. No obstante, esta es una de las fuentes que va utilizar el proyecto. Lo que se propone es una mejora a los datos que proporcionan estas aplicaciones para poder extraer un significado correcto y sacar conclusiones de una forma más eficiente y precisa.

El proyecto tiene mucho valor al automatizar un proceso que puede llegar a ser muy complejo de hacer manualmente. Además propone una muy buena solución a un problema que es muy común y puede llegar a proporcionar información muy valiosa sobre el estado de un deportista.

La complejidad es causada porque manualmente requiere analizar información de semanas de entrenamientos de cientos de atletas. Al final esta tarea no se acaba realizando por completo o se dedica muy poco tiempo a ella con un par de comprobaciones de gráficas que hacen llegar a conclusiones con poca exactitud.

Construyendo una funcionalidad de análisis automático se podría explotar computacionalmente toda esta información para dar respuesta al estado de forma actual del deportista y como va evolucionando a lo largo del tiempo de una forma rápida, eficiente, exacta y sobretodo interesa que sea escalable.

3. Alcance

3.1 Objetivos

El objetivo global del proyecto es analizar de forma automática un conjunto de entrenamientos de un deportista para poder evaluar su estado de forma.

El primer paso es hacer un análisis en profundidad de un subconjunto de los datos, en concreto de un número reducido de parámetros de rendimiento, para luego poder escalar. Un parámetro es un valor numérico que se puede extraer a partir de diferentes datos del dispositivo.

Uno de los objetivos principales es analizar los datos para conocer sus características más importantes, la cantidad de valores perdidos y *outliers*. A partir de aquí, se construirán métodos de preprocesamiento como filtros para obtener unos datos más limpios y adecuados.

Es importante remarcar que muchas veces interesa realizar observaciones sobre un conjunto de entrenamientos y no suele ser tan importante la información de uno en concreto para poder determinar el estado de forma de un deportista. Por lo tanto, la aproximación que se ha realizado consiste en determinar la evolución de los parámetros de rendimiento registrados durante meses y predecir la tendencia. Generar una predicción a partir de datos es un problema computacional que el proyecto intenta resolver mediante aprendizaje supervisado.

El objetivo principal más importante es estudiar, aplicar y evaluar modelos de aprendizaje supervisado sobre parámetros de rendimiento de un deportista para predecir la tendencia en un futuro.

Por último, las funcionalidades del proyecto serán implementadas en la aplicación web de Trainerer para poder ser utilizadas por sus clientes.

3.2 Posibles obstáculos y riesgos

Uno de los posibles obstáculos que puede surgir durante el desarrollo del proyecto puede ser el acceso a la información del dispositivo del deportista. En breves palabras, esta información se distribuye en una serie de datos generales y un conjunto de sectores. Estos sectores contienen una estructura de datos que proporciona información por cada segundo del entreno que contiene la latitud, la longitud, la elevación o altitud, el pulso y la velocidad, entre otros. Esta estructura de datos tendrá el nombre de Punto.

De esta forma se obtiene una cantidad muy elevada de datos a analizar.

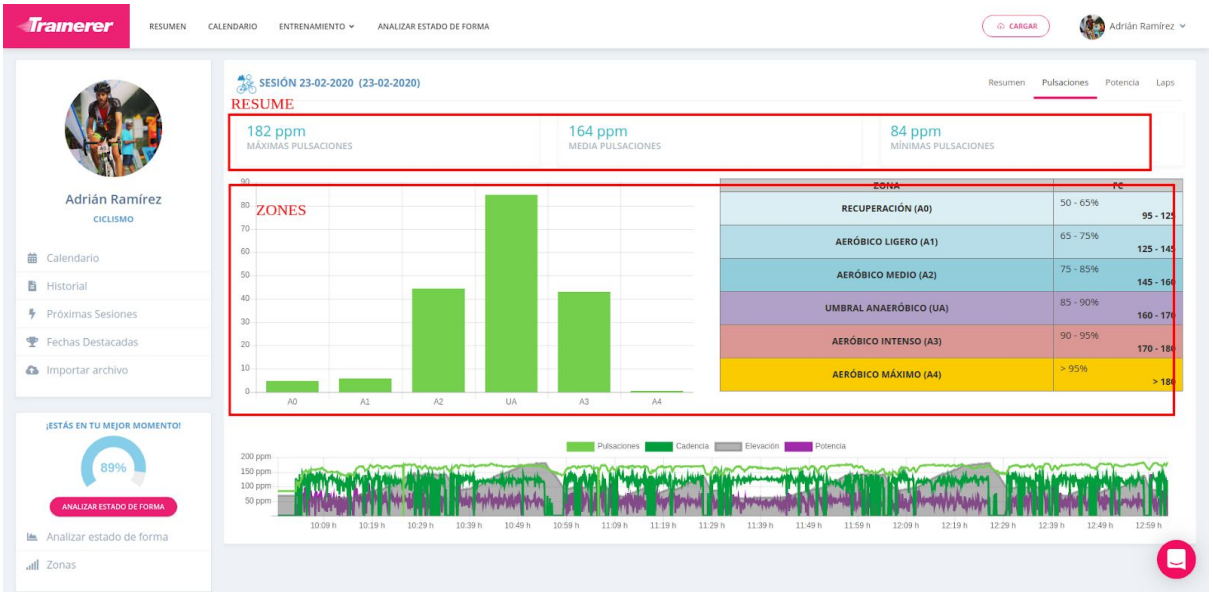


Figura 1. Panel de la aplicación web de Trainerer donde se muestra una gráfica con todos los puntos y valores que se obtienen durante un entrenamiento del ciclista Adrián Ramírez.

La funcionalidad recibirá un preproceso de esta información en forma de parámetros para no acceder a una estructura tan grande. El problema que se puede generar es que si en el preproceso no se proporciona la información suficiente es muy costoso computacionalmente acceder a los puntos para realizar el cálculo necesario. Estos puntos podrían ocupar mucho espacio en memoria y hacer muy ineficiente el análisis ya que son un gran cantidad y aumentan por cada atleta y entrenamiento.

Una solución a este problema sería organizar la base de datos de forma diferente para poder tener un acceso rápido y menos costoso en memoria. No obstante, cambiar la estructura haría que la base de datos ocupara mucho más espacio cosa que no elimina el problema.

En definitiva, se tendrá que hacer casi todo lo posible para evitar este tipo de accesos.

3.3 Metodología

Es necesario definir un conjunto de métodos que se utilizan durante la investigación para organizar su desarrollo. Un ejemplo son las herramientas utilizadas para hacer un control de seguimiento y los pasos a seguir durante su avance.

Durante todo el desarrollo del proyecto se seguirá una forma de trabajar iterativa. Es decir, primero se analizará el estado del proyecto y se detectarán sus defectos, luego se planificaran las posibles mejoras y soluciones a algunos problemas, se implementarán y finalmente se volverá a validar el estado para ver el progreso realizado. Esta metodología se volverá a repetir hasta conseguir los objetivos finales del proyecto.

Esta forma de trabajar se verá consolidada con una constante supervisión por parte del director del proyecto ya que es el guía en su desarrollo. Durante el trabajo se intercambiarán ideas para el avance del proyecto y se expondrán antes de continuar para poder mejorar y corregir errores. Además, otro compañero que está realizando en Trainerer otro proyecto sobre una funcionalidad de planificación de entrenamientos también aportará ideas para generar un beneficio mutuo.

La herramienta que se usará para realizar un control de versiones será *git* con ayuda de una interfaz gráfica que será GitLab [3]. Así se facilitará la disponibilidad del código y la recuperación de versiones anteriores en el caso de producirse un error crítico que pueda afectar gravemente al proyecto. Se usará Google Drive para documentar los resultados y preparar la documentación así como el curso de GEP. Para la comunicación interna dentro de la empresa y con el director se usará Slack [4] dónde se compartirán archivos y se especificarán fechas para hacer reuniones de seguimiento, entre otros.

Los modelos de aprendizaje supervisado en ningún caso se implementarán desde cero. Esta tarea sería muy costosa, demasiado específica y concuerda muy poco con los objetivos del proyecto. En su lugar se han utilizado las funciones disponibles en la librería de Python Scikit-Learn [5] que es de código abierto.

Por último, el proyecto se ha estructurado en formato de un paquete de Python para que desde las diferentes partes del proyecto se pueda acceder a las funciones de código principales. También se ha usado la programación orientada a objetos y el patrón de diseño llamado *Repositorio*. Este patrón define como modelo una clase de Python que define un objeto; un repositorio como un conjunto de funciones que acceden a bases de datos para consultar, guardar y actualizar objetos; y un servicio como funciones asociadas a un objeto en concreto.

4. Planificación temporal

4.1 Duración del proyecto

La fecha de inicio del proyecto es el 15 de enero de 2020 y la fecha de finalización es el 15 de junio de 2020. La duración global será de 400 horas y la fecha prevista para la lectura del TFG es el 2 de julio. El investigador trabajará 4 horas de lunes a viernes realizando un total de 20 horas semanales.

A continuación se muestra una descripción de las tareas y una estimación temporal hecha al iniciar al proyecto.

4.2 Descripción de las tareas

- **Gestión del proyecto**

Esta tarea va vinculada al curso de GEP, dentro del marco del trabajo de fin de grado. En este curso se enseña a cómo planificar y distribuir las tareas amén de preparar la documentación y evaluar su calidad. Los puntos que se tratan durante el curso son los siguientes:

- Contextualización y alcance
- Planificación temporal
- Presupuesto y sostenibilidad
- Evaluación final
- Herramientas TIC de soporte a la gestión de proyectos y equipos

En cada una de ellas ha sido utilizado Google Drive para su redacción, Gantter [6] para generar el diagrama de Gantt y Atenea para hacer entregas al profesor del curso.

- **Diseño**

En el proyecto será necesario un conocimiento avanzado de Python, de bases de datos no relacionales y relacionales, en concreto MongoDB [7] y MariaDB [8] respectivamente; más un agente de mensajes que se llama RabbitMQ [9]. En esta primera etapa es necesario familiarizarse con estas herramientas y los conceptos en que se basan para ser capaz de utilizarlas con total conocimiento.

Seguidamente, se procederá a realizar un diseño del software donde las funcionalidades serán implementadas. Será necesario realizar relaciones entre los diversos agentes así como un esquema general.

Durante la implementación se utilizará Visual Studio [10], un IDE que ayudará a gestionar todos los ficheros de código. También se usarán diversos *frameworks* como Robo3T [11] para hacer una gestión rápida y eficiente de las bases de datos. Para

comunicarse con el director del trabajo y el resto de la empresa, se utilizará la plataforma de mensajería Slack como vía de comunicación.

- **Implementación de un ODM y un ORM**

Como se explica en la metodología, hará falta una implementación de un sistema intermedio que permita tratar como clases los elementos de las bases de datos para un uso más práctico y que ayudará en el proceso de implementación. La finalidad de este paso será implementar una capa que facilitara muchos accesos.

- **Implementación de récords**

Esta es una de las funcionalidades que ofrecerá el proyecto y que dará información general para poder ser mostrada en la aplicación web de Trainerer.

Como todo proceso de implementación se harán los tests de validación y los tests de rendimiento. En los tests de validación se va a comprobar que el código es correcto y pasa todos los juegos de prueba, por otro lado los tests de rendimiento ayudarán a consolidar y dar el visto bueno a la eficiencia del código.

Se utilizarán las herramientas de la primera fase añadiendo Google Drive para documentar los resultados que se vayan obteniendo.

- **Predicción de tendencia**

Esta es la funcionalidad más avanzada ya que va a requerir de bastante horas de observación de datos y gráficas para poder determinar los modelos de aprendizaje automático más adecuados a los datos.

Se utilizarán las herramientas de las fases anteriores juntamente con librerías como Matplotlib [12] para poder hacer una visualización de datos correcta en forma de gráficas.

Para realizar análisis estadísticos en detalle se utilizará la aplicación web Jupyter Notebook [13].

- **Evaluación rendimiento**

Una vez implementada todas las funcionalidades anteriores, se debe explorar el código en busca de errores y probar su escalabilidad. Puede darse el problema que el código no funcione y sea muy lento cuando se pruebe para bastantes instancias de deportistas. En este caso, se pasará a hacer una refactorización y solventar el problema para hacerlo más rápido.

- **Hito final del proyecto**

En esta última tarea se documentará todo el proyecto. Se preparará la presentación oral ante un tribunal de la especialidad y se utilizarán los diferentes servicios de Google Drive para la documentación de la memoria y las diapositivas para la presentación oral.

4.2 Estimación temporal

Una vez definidas las tareas y subtareas en las que se va realizar el trabajo, se debe estimar el tiempo que consumirá y precisar sus dependencias. En el Anexo 1 se puede ver el diagrama de Gantt correspondiente. En la siguiente tabla se resume con detalle:

Id.	Descripción	Duración aprox.	Depende de
1	**Gestión del proyecto**	**80h**	
2	Contextualización y alcance	20h	
3	Planificación temporal	10h	
4	Presupuesto y sostenibilidad	10h	3
5	Herramientas TIC	10h	
6	Evaluación final	30h	4
7	**Diseño**	**40h**	
8	Implementación de un ODM y ORM	10h	
9	**Implementación de récords**	**60h**	7
10	Récords	20h	
11	Aumentos	40h	
12	**Predicción de tendencia**	**130h**	
13	Extracción de datos	20h	
14	Análisis	60h	
15	Implementar modelos de aprendizaje automático	50h	14
16	**Evaluación rendimiento**	**30h**	15

17	**Hito final**	**60h**	
18	Documentación	10h	
19	Preparación defensa	50h	18

Tabla 1. Tabla de tareas y subtareas de la planificación.

Se estima una duración total de 400 horas.

4.2.1 Dependencias entre tareas

- ➔ **Evaluación final** (5) depende de los entregables anteriores ya que es una unificación de estos teniendo en cuenta los consejos y correcciones dadas por los profesores de GEP.
- ➔ **Implementación de récords** (9) depende de **Diseño** (7) porque primero hay que hacer un breve estudio de lo que se pretende elaborar y no hacer un diseño trivial que genere errores durante el desarrollo.
- ➔ **Implementar modelos de aprendizaje automático** (15) depende de **Análisis** (14) porque primero se tiene que realizar un análisis de una gran cantidad de entrenamientos de deportistas para poder observar patrones que siguen los datos e intentar determinar el modelo que más se ajusta con exactitud de forma general.

4.2.2 Concurrencia entre tareas

Diseño con **Evaluación final** - Mientras se va progresando en el curso de GEP y solo falta preparar la entrega final la fase de diseño puede empezarse ya que no depende de ninguna parte de GEP.

Récords con **Extracción de datos** - La parte de implementación de aumentos se puede implementar con una serie de datos artificiales que simulen actividades de deportistas, pero no es necesario experimentar con datos reales. Es por eso que a la vez que se hace una extracción de estos datos reales se puede implementar sin ningún problema la funcionalidad de récords de los parámetros.

Evaluación rendimiento con **Documentación** - Se puede empezar a preparar la primera parte de la documentación empezando por el principio del proyecto mientras se evalúa el rendimiento de la implementación hecha. Aunque ocurra algún problema que obligue a modificar en gran parte la implementación no afectará a toda la documentación ya que estará reflejado en su parte final.

4.3 Recursos

A continuación se hará una lista de los recursos necesarios para desarrollar el proyecto.

4.3.1 Recursos humanos

El investigador es el principal responsable del proyecto y el que se encargará principalmente de su desarrollo. También entrarán en juego los profesores de GEP, el ponente, el director y otros compañeros de la empresa que tendrán un rol de soporte adicional.

4.3.2 Recursos hardware

- **Ordenador portátil Medion Erazer**: 7,7GB de DRAM y procesador Intel Core i7-8550U 1,8 GHz
- **Ordenador Gigabyte Brix**: 16 GB de DRAM y procesador Intel Core Quad core i5-8250U

4.3.3 Recursos software

- **Docker**: herramienta diseñada para facilitar la creación, desarrollo y ejecución de aplicaciones usando contenedores.
- **Visual Studio**: IDE de proyectos desarrollado por Microsoft que soporta el lenguaje Python.
- **Jupyter Notebook**: aplicación web de código abierto para crear y compartir documentos que contienen código interactivo, visualizaciones y texto.
- **Robo3T**: interfaz gráfica para MongoDB de código abierto.
- **Google Drive**: documentación de los resultados durante el proyecto, las diapositivas de la presentación oral y los entregables de GEP.
- **Gitlab**: interfaz de Git para controlar las versiones del código y poder recuperarlas.
- **Slack**: aplicación de mensajería entre el director del trabajo, la empresa y el investigador.
- **Ubuntu 18.04**: sistema operativo del portátil y el ordenador.
- **Atenea**: página web que se utiliza para la comunicación con los profesores de la asignatura de GEP.
- **Webmail FIB y Gmail**: servicios de correo electrónico para comunicarse con el ponente.

4.4 Gestión del riesgo

Suele suceder que durante el transcurso de la mayoría de proyectos pueden presentarse obstáculos que impidan su desarrollo eficaz. Muchos de los riesgos ya han sido identificados en el apartado de posibles obstáculos y se ha propuesto una solución. Es por eso que la parte de predicción es el conjunto de tareas con más horas designadas porque al tratarse de una tarea de análisis de datos siempre pueden salir imprevistos y es necesario tener un margen de tiempo para poder tratar sin problemas esos inconvenientes.

Además hay trabajo que se puede realizar concurrentemente de forma que se puede avanzar de otras formas si se produce alguna complicación en alguna tarea. Ante todo se respetará el método de trabajo que se definió en Metodología que obliga a evaluar un pequeño avance en la implementación antes de seguir avanzando. Así se ofrece una buena herramienta para sortear los posibles obstáculos que surjan durante esta fase que es la más larga y compleja.

Para acabar, la supervisión del director será necesaria para ayudar si surgen complicaciones con la complementación del ponente que aconsejara más sobre la parte técnica.

5. Identificación y estimación de los costes

En esta sección viene explicada en detalle la gestión económica del proyecto.

5.1 Presupuesto

Los costes se han catalogado según los siguientes recursos: humanos, software y hardware. Además de estos costes, también hay que contar con costes indirectos como gastos de oficina para luego poder elaborar un presupuesto final que vendrá desglosado según las tareas de la planificación elaborada.

En cada uno de los apartados se detallarán las amortizaciones, imprevistos, contingencias y costes calculados. Puede darse el caso que se tenga que describir mecanismos para controlar las desviaciones en relación al presupuesto que irán junto a indicadores numéricos para hacer un control más simple.

5.1.1 Costes humanos

Una manera de detallar el coste de los recursos humanos es según las ocupaciones que se necesitan y la cantidad de horas que se dedican.

Como hemos mencionado antes los sujetos implicados son: el investigador, como desarrollador junior; el director, el ponente y el profesor de GEP, como gestores del proyecto y por último el CTO de la empresa que es un desarrollador de software senior. Ahora pasamos a la fase de estimación de su salario con la siguiente pa ǵ ina web [14]:

https://www.payscale.com/research/ES/Country=Spain/Salary

Rol	Coste por cada hora (€/h)
Ingeniero de Investigación junior	9,00
Gestor de proyectos	40,00
Desarrollador de software senior	60,00

Tabla 2. Tabla indicando el coste por hora de cada uno de los roles.

Ahora se pasa a calcular las horas dedicadas por cada rol en cada una de las tareas:

Tarea	Horas(h)	Rol		
		Ing. junior	Gestor	Des. senior
Gestión del proyecto	**80**	50	30	-
Diseño	**40**	35	-	5
Implementación de récords	**60**	50	5	5
Predicción de tendencia	**130**	100	20	10
Evaluación rendimiento	**30**	25	5	-
Hito final	**60**	55	5	-
Total	**400**	315	65	20

Tabla 3. Tabla de horas de cada rol por tarea.

El presupuesto final es:

Rol	Horas(h)	Coste(€)
Ingeniero de investigación junior	315	2835,00
Gestor de proyectos	65	2600,00
Desarrollador de software senior	20	1200,00
Total	**400**	**6635 €**

Tabla 4. Tabla del total de horas y coste de los recursos humanos.

Como los imprevistos afectan a la cantidad de horas que se dedican implica que en recursos humanos es donde puede haber más desviación económica, sobretodo en la parte del ingeniero de investigación junior y los gestores de proyectos (director y ponente).

5.1.2 Costes software

Todas las herramientas y programas que se utilizan en este proyecto son gratuitos o se han utilizado las versiones gratuitas. Por lo tanto, no habrá ningún tipo de desviación que provenga de esta parte.

5.1.3 Costes hardware

A continuación se exponen los precios de los recursos hardware descritos en su apartado correspondiente a más de algunos secundarios no contemplados.

Hardware	Coste(€)	Vida útil(años)	Amortización(€)
Ordenador portátil Medion Erazer	600,00	4	200,00
Ordenador Gigabyte Brix	460,95	4	115,30
Teclado Logitech	30,00	4	7,50
Ratón Logitech	10,00	4	2,50
Pantalla Dell x2	430,00(215,00)	4	107,50(53,75)
Total	**1530,95 €**	**-**	**432,80 €**

Tabla 5. Tabla de costes hardware.

A priori no debería de haber variabilidad en el coste de los recursos hardware. Si el ordenador dejase de funcionar se puede sustituir fácilmente por otro propiedad de la empresa. No obstante, si el portátil se estropease se podría utilizar uno de coste más barato pero que igualmente se debería tener en cuenta.

5.1.4 Costes indirectos

Estos gastos implican los gatos de oficina que está ubicada en un espacio de *coworking* llamado Canòdrom Parc de Recerca Creativa.

Recurso	Coste(€)
Oficina	720

Tabla 6. Tabla de costes indirectos.

Este número se obtiene de los 180 euros al mes que la empresa paga a la empresa que gestiona el espacio por tener una silla dentro de la oficina que utilizará el investigador.

5.2 Control de gestión

Como antes se ha comentado, este proyecto podría sufrir desviaciones en la planificación que afectarían directamente al presupuesto. En los anteriores apartados se han descrito diversas formas para evitar estas posibles desviaciones pero no exime de no tenerlas en cuenta.

El presupuesto de recursos humanos es el que tiene más posibilidades de verse afectado. Para elaborarlo se ha utilizado una aproximación reflejada en el apartado de planificación temporal que podría tener bastantes desviaciones durante el desarrollo del proyecto. Si la duración de una tarea cambia afecta directamente a las otras en duración de horas y por lo tanto también de precio. Las horas asignadas al investigador y a los gestores de proyecto son las más inestables por lo tanto el presupuesto de imprevistos se calculará acorde a ellos.

Por otro lado, hay una probabilidad casi nula de que los costes de hardware se vean alterados. Podría suceder un daño físico o virus que dañase estos elementos pero no implican una desviación en horas de una tarea.

El presupuesto de software como se ha observado es el más económico y el único caso en que se pudiera incrementar es si se acaban utilizando softwares de pago que en principio no van a ser necesarios en absoluto.

En cuanto a los costes indirectos podrían variar si la cantidad total de horas del investigador cambia ya que implicaría modificar la cantidad de horas presenciales en la oficina.

Actividad	Coste(€)	Porcentaje de riesgo	Observaciones
Contingencia	1327,00	-	Margen de seguridad, calculado como un porcentaje de 20% en base al presupuesto de recursos humanos
Imprevistos	-	-	No se muestran en el diagrama de Gantt. Se especifica el tipo de riesgo, el coste de la alternativa y la probabilidad

Virus, error o robo en el portátil (Coste alternativa: 500€)	25,00	5%	La alternativa es utilizar un ordenador portátil de menor calidad, el coste sería su importe
Virus, error o robo en el ordenador (Coste 460,95)	46,00	10%	La alternativa es utilizar otro ordenador igual de la empresa
Total	**1398,00 €**	**-**	**-**

Tabla 7. Tabla de gastos de imprevistos y contingencia.

5.3 Presupuesto final

Una vez se han tenido en cuenta todos los costes por separado se junta todo para obtener un presupuesto final para el proyecto:

Recurso	Coste(€)
Recursos humanos	6.635,00
Recursos hardware	1.530,95
Recursos software	0,00
Costes indirectos	720,00
Imprevistos y contingencia	1.398,00
Total	10.283,95
Total con IVA	**12.443,58€**

Tabla 8. Tabla con el presupuesto final del proyecto.

6. Diseño

En este apartado se explica como se ha llevado a cabo el diseño del trabajo.

6.1 Definición de parámetros y datos

6.1.1 Definición de zonas

Para poder implementar un modelo que analice un entrenamiento de un deportista es necesario definir unos factores de rendimiento.

Primero de todo hace falta tener un control de las zonas de intensidad que el deportista puede alcanzar durante el entreno. Esta zona puede estar influenciada por el pulso o bien por la potencia recogida en el dispositivo de la bicicleta creando así una distinción en dos categorías. Este dispositivo es opcional y sólo lo tienen una parte de los ciclistas. Realizando un preprocesamiento se recogen los intervalos totales de tiempo que el deportista ha estado en cada zona y dentro de estas se definen tres parámetros de rendimiento.

Estos parámetros son **resistencia**, tiempo total que está dentro de la zona (s); **velocidad**, velocidad media obtenida (m/s) y **fuerza**, potencia media obtenida (W).

No obstante, para poder considerar que un deportista se está ejercitando dentro de una zona de intensidad se tiene que cumplir un tiempo mínimo consecutivo.

Zona	Pulso	Potencia	Tiempo mínimo consecutivo
A2	75% - 85%	75% - 90%	10’
UA	85% - 90%	90% - 105%	1’
A3	90% - 95%	105% - 120%	30’’
A4	> 95%	> 120%	10’’

Tabla 9. Tabla de tipos de zona con sus límites en porcentaje y tiempo mínimo consecutivo.

Como podemos observar el tipo de zona relacionada con el pulso está definida dentro de un porcentaje sobre el total de pulsaciones máximo de un deportista en concreto, por lo que los límites de las zonas no son iguales para todos. Por otro lado, el porcentaje de potencia está definido sobre la máxima que se puede mantener en una hora que se llama FTP (por sus siglas en inglés, *Functional Threshold Power*) [15], también distinta para cada deportista. Cabe destacar que en la zona UA se acepta un margen de error de 30 segundos si el deportista sale de la zona pero vuelve a entrar en este pequeño intervalo.

Para las funcionalidades que se usarán en este proyecto y simplificar el problema solo se analizará la zona UA y superiores como si fueran un conjunto, por lo tanto se considerará que los factores de rendimiento de un deportista son los que se consiguen en estas zonas.

Dentro del preproceso de los datos, si se detecta que un atleta ha estado dentro de la zona UA o superiores más de una vez durante un entrenamiento, cumpliendo la duración mínima de un minuto, se tendrá en cuenta el intervalo con el mayor tiempo consecutivo y los demás se descartaran. Por lo tanto, los parámetros de rendimiento estarán asociados únicamente a este intervalo seleccionado.

6.1.2 Definición de intervalos y parámetros

Dentro de los muchos valores que se pueden extraer de una actividad, hay algunos que es importante tener controlados:

- **Pulso:** Frecuencia cardíaca media por minuto (bpm)
- **Velocidad (m/s)**
- **Cadencia:** Revoluciones por minuto que dan las bielas de la bicicleta al pedalear (rpm) [16]
- **Potencia:** Potencia media generada en el pedaleo durante un período de tiempo (W) [17]
- **Pendiente:** Pendiente media positiva del terreno (%) [18]

Los dos últimos sólo están disponibles en algunas sesiones de ciclismo. Durante el preprocesamiento de la información se generan cálculos de los valores mínimos y máximos de la media de estos parámetros en todo el conjunto de la actividad para intervalos de tiempo de 5 segundos, 1 minuto, 20 minutos y una hora. Por ejemplo, si un deportista obtiene un mínimo de 108,3 W y un máximo de 111,9 W de potencia en un intervalo de 1 hora significa que para todos los intervalos consecutivos de 1 hora del entreno, el máximo obtenido de todas las medias de potencia es 111,9 W y el mínimo obtenido es 108,3 W.

Estos parámetros estarán en todo momento calculados dentro de la base de datos, por lo que el proyecto podrá leerlos siempre exceptuando cuando no estén definidos por causa de errores en la API.

6.2 Fase previa

6.2.1 Planteamiento inicial

Antes de empezar, hay que entender el entorno y aprender los conceptos básicos de las herramientas donde se desarrollará el proyecto. Como se ha mencionado consistirá en varias funcionalidades que estarán implementadas en una plataforma más grande que es una aplicación web. A continuación se describen las partes que se comunicarán con el proyecto directamente y algunos de los recursos que consumirá.

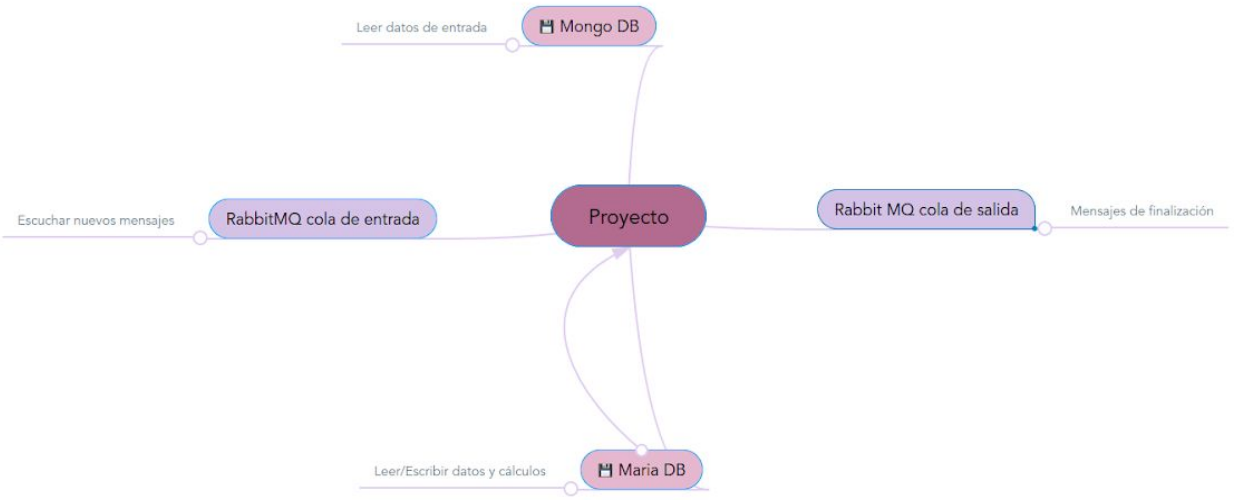


Figura 2. Esquema del entorno del proyecto.

En este esquema podemos ver las diferentes unidades con las que el proyecto establecerá comunicación. Por un lado el proyecto deberá escuchar mensajes de una cola en RabbitMQ. En esta cola de entrada escribirán otros procesos que soliciten un análisis o una de las funcionalidades del proyecto y cuando estén los datos de entrada listos para ser leídos en MongoDB. El proyecto escuchará está cola y cuando reciba un mensaje ejecutará una funcionalidad.

MongoDB se usará, entre otras cosas, para almacenar los datos de entrada que el proyecto necesitará y tendrán forma de documentos (diccionarios). MariaDB se usará como proceso intermedio, las funcionalidades escribirán los datos que precisen para en ejecuciones futuras tenerlos disponibles y poder explotarlos. Además se guardarán valores asociados a parámetros y constantes para evitar definirlos dentro del código, dando así escalabilidad al proyecto y permitiendo su modificación de manera externa.

Finalmente tenemos otra vez RabbitMQ pero de forma inversa. Cuando se acabe de ejecutar la funcionalidad solicitada y estén listos los resultados, se escribirá una mensaje en esta cola para que sea escuchado por otros procesos que podrán usar la salida generada por el proyecto.

6.2.2 Docker

Teniendo en cuenta lo explicado en la idea inicial, ahora se introducirá en qué consiste Docker y qué papel desarrollará en el proyecto.

Docker es un proyecto *open-source* popular basado en contenedores Linux [19]. Usando características del Kernel de Linux como *namespaces* y grupos de control crea contenedores encima de un sistema operativo. Además contiene componentes bien definidos que hacen que el empaquetamiento sea más fácil.

Permite separar diferentes aplicaciones de la infraestructura del proyecto en el que se aplica de forma que el *software* se puede entregar más rápidamente. Tiene herramientas que permiten el envío, testeo y despliegue de código en poco tiempo de forma que reduce significativamente el retraso entre escribir código y desplegarlo.

Esto último es lo más importante para el proyecto ya que será ejecutado dentro de un contenedor Docker con el objetivo de facilitar el testeo y un despliegue futuro dentro de la aplicación de Trainerer como se ha explicado.

Las bases de datos y RabbitMQ estarán implementados en su correspondiente contenedor del cual dependerá el del proyecto.

6.2.3 Modificación

Debido a las causas explicadas en el informe de seguimiento, se tomó la decisión de dejar a un lado parte de la idea inicial y llevar a cabo una alternativa para mostrar los resultados obtenidos y ejecutar las funcionalidades que se quieren implementar.

Primero de todo se eliminará la dependencia con la plataforma y las comunicaciones a través de RabbitMQ. No obstante se mantendrán las lecturas y escrituras con las bases de datos con la excepción de MariaDB. La estructura de contenedores de Docker también seguirá en uso.

El elemento sustituto que se ha utilizado es un *framework* web llamado Flask [20]. Consiste en un conjunto de herramientas, librerías y tecnologías que permiten construir una aplicación web.

Es por eso que se construirá una aplicación en forma de *dashboard* que se ejecutará en local para poder testear y ejecutar de forma más sencilla las diferentes funcionalidades implementadas. De alguna forma, intentará hacer de sustituto de la aplicación web de Trainerer y eliminará la tarea de desplegar las funcionalidades. Además proporciona una alternativa para no presentar resultados en consola y así darle un aspecto más estético y robusto al proyecto.

Se ha elegido Flask porque está orientado a microservicios, ya que en la idea inicial las funcionalidades pretenden ser integradas de esta manera. Este concepto implica que no suele tener dependencias en librerías externas y tiene varias ventajas y desventajas. Por un lado, el *framework* es más ligero con pocas dependencias que actualizar y menos a vigilar en errores de seguridad. Por otro lado, a veces será necesario utilizar *plugins* y realizar más trabajo. En el caso particular del proyecto, no se utilizarán *plugins* y la mayoría de dependencias que existirán serán internas.

Cabe destacar que Flask está construido sobre dos herramientas:

- Werkzeug, una librería de utilidades WSGI
- Jinja, un motor de plantillas (*templates*)

6.3 Arquitectura y diagrama de clases

6.3.1 ODM y ORM

En el apartado anterior se ha mencionado que se realizarán accesos a dos tipos de bases de datos. Una base de datos relacional, llamada MariaDB (aunque no es utilizada) y otra no relacional llamada MongoDB. Siguiendo unas buenas prácticas en este proyecto se usan dos patrones de programación que controlan los accesos a los datos: ODM para MongoDB y ORM para MariaDB.

Un *Object Document Mapper* (ODM) [21] es un módulo que sirve para tratar documentos de una base de datos no relacional como objetos. Es muy recomendable en algunos casos no hacer una programación desde cero y ayudarse de librerías que solucionan un problema parecido. Es por eso que para implementar el ODM se ha utilizado HumbleDB [22] que es un contenedor construido alrededor de la librería Pymongo.

Por otro lado, un *Object Relational Mapping* (ORM) hace la misma función que un ODM pero con bases de datos relacionales como MySQL o MariaDB. En este caso el objeto que se crea se aplica a un modelo relacional que contiene datos en un formato incompatible para el proyecto que se acaban convirtiendo en tipos de un lenguaje de programación orientado a objetos (Python). Un ORM recoge los detalles específicos de implementación de los *drivers* de almacenamiento en una API (*application program interface*) y convierte los campos relacionales en objetos miembro. Como en el caso anterior, se utilizará la librería llamada SQLAlchemy [23] para implementar este patrón.

6.3.2 Diagrama de clases

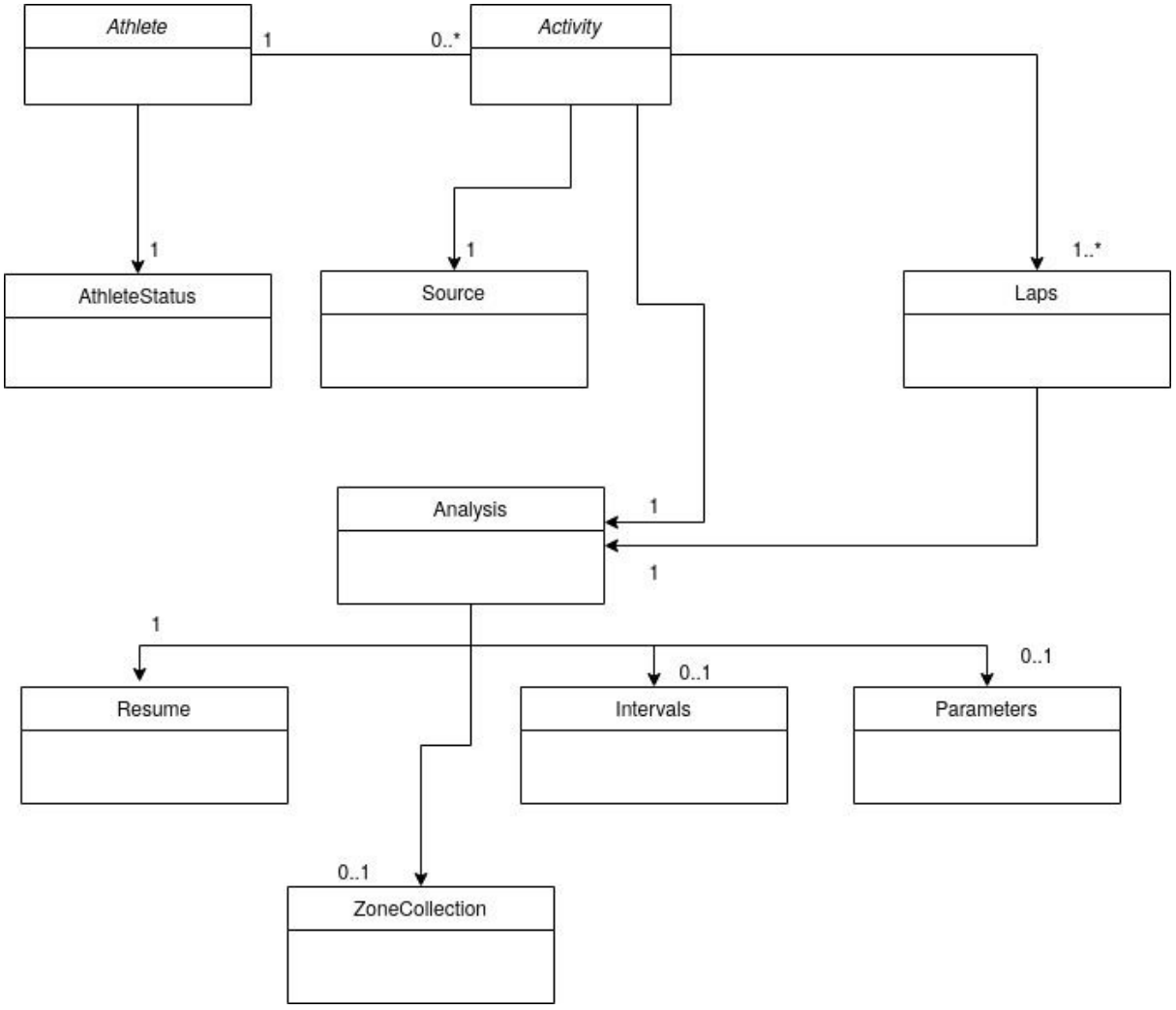


Figura 3. Diagrama de clases del proyecto.

6.3.3 Descripción de clases

- **Activity**: Representa un entrenamiento almacenado en base de datos relacionado con un deportista. Contiene a todas las demás clases y es de donde se extrae toda la información necesaria para el proyecto.
- **Athlete:** Representa un deportista que está almacenado en base de datos. Dispone de un conjunto de actividades que ha realizado que puede ser vacío.
- **AthleteStatus**: Contiene campos de información relativa al deportista como su número identificador, género, edad, peso, altura y medidas de rendimiento como su máximas pulsaciones y su FTP.
- **Source**: Información sobre el dispositivo que ha registrado los datos de la actividad.
- **Analysis**: Información relativa a parámetros y valores de referencia que el atleta ha conseguido durante la actividad.
- **Laps**: Segmentos en el que el deportista ha escogido dividir la sesión con ayuda de su dispositivo. Puede ser que la actividad no tenga segmentos y se considera que toda la actividad es uno y puede llegar a ser tantos como el deportista escoja sin ningún límite. Cada *lap* tiene asociado un *analysis* que contiene lo mismo que para la *activity* pero asociado a ese *lap* específico.
- **Resume**: Parámetros generales de la actividad o segmento como es la distancia total, la duración, el desnivel y las calorías consumidas entre otros.
- **Intervals**: Lista de parámetros relacionados con intervalos de tiempo con su media mínima y máxima para cada intervalo en el total de la actividad.
- **ZoneCollection**: Conjunto de clases Zone.
- **Zone:** Zonas de rendimiento relacionadas con el pulso. Cada zona tiene los porcentajes en los que está definida, cuánto tiempo total del entreno se ha estado en esta zona y los parámetros: velocidad, resistencia, fuerza y pendiente. Aunque puede suceder que no estén definidos.
- **Parameters**: Resumen de los valores de los parámetros relacionados con intervalos de tiempo con valor mínimo, medio y máximo total en toda la actividad.

7. Análisis de datos

En un proyecto en el que es necesario realizar predicciones es muy importante conocer la forma, las características y las propiedades de los datos. El motivo más importante es determinar si es un problema que tiene solución, ya que si se observa que los datos tienen un gran factor aleatorio seguramente no funcionaria ningún modelo predictivo. Este será el principal objetivo del análisis, ver los datos en detalle a través de gráficas para ver si siguen algún patrón o tienen alguna periodicidad. A más se observará si los datos presentan mucho ruido o poco y si hay muchos *outliers* a tratar o valores perdidos.

Después de un análisis visual a través de gráficas, es esencial formalizarlo y realizar un análisis estadístico. En el Anexo 2 se muestra cómo se ha realizado utilizando Jupyter Notebook y las conclusiones que se han obtenido.

Es importante mencionar que para simplificar el problema el único deporte que se analizará en detalle por mayoría en cantidad de datos y con el que se probarán funcionalidades será *CYCLING_STREET*. Este nombre es su identificador en la base de datos y se refiere a ciclismo en ruta.

A continuación, se mostrarán varias gráficas que marcan un patrón definido en el parámetro *avgSpeedMetersPerSecond*.

Las siguientes gráficas corresponden a actividades del atleta con identificador número 292 en base de datos.

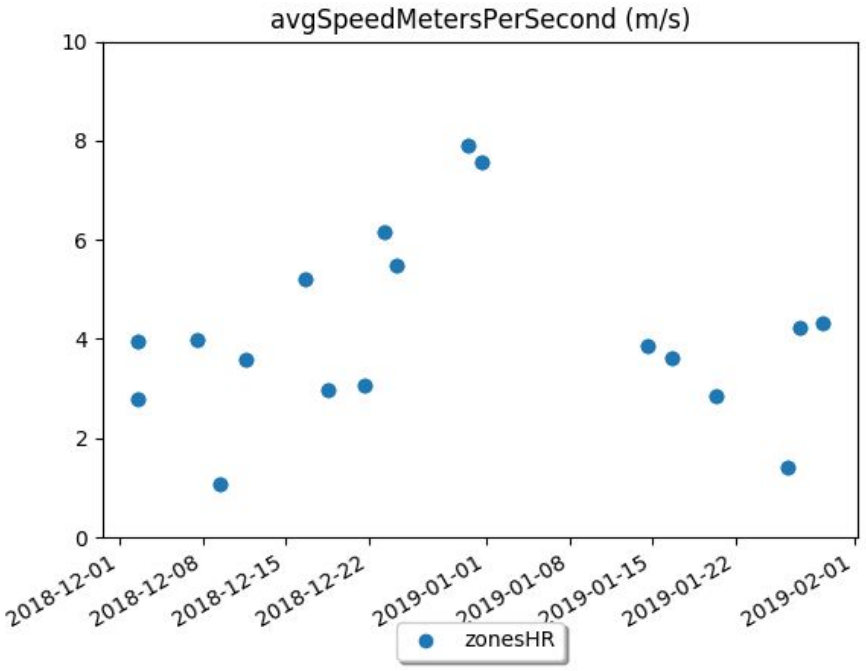


Figura 4. Gráfica de actividades del 1 de diciembre de 2018 hasta el 1 de febrero de 2019.

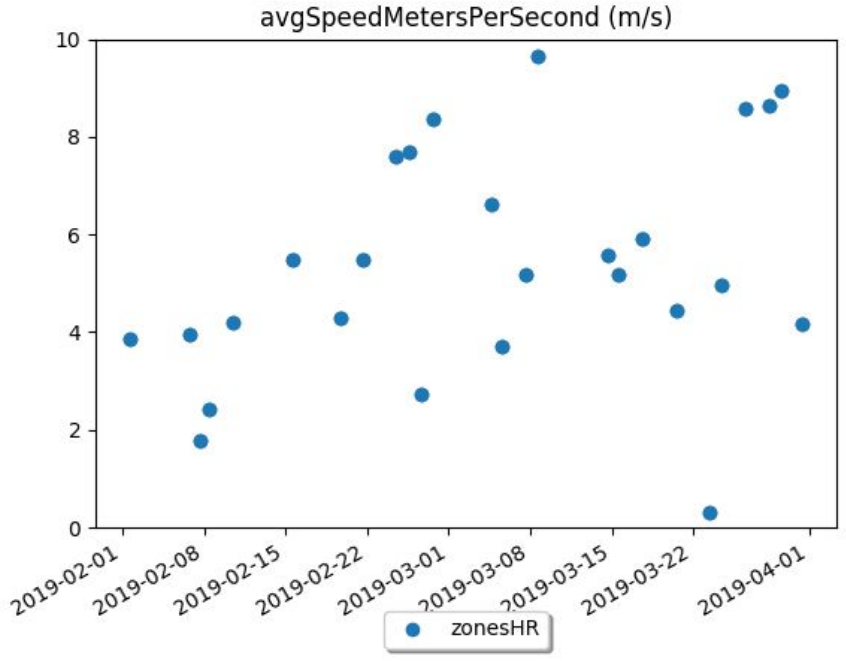


Figura 5. Gráfica de actividades del 1 de febrero de 2019 hasta el 1 de abril de 2019.

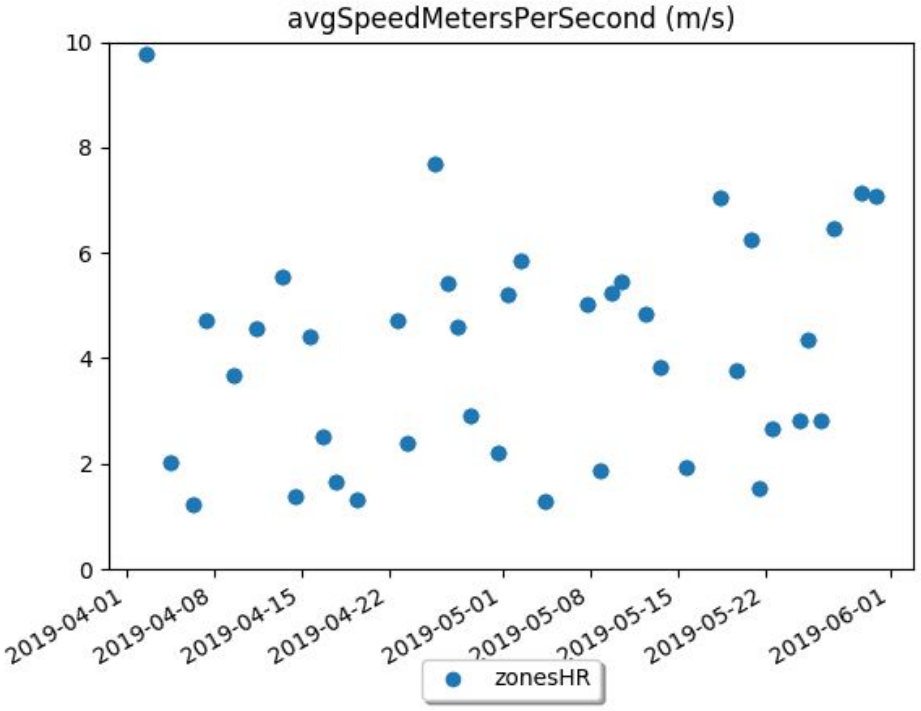


Figura 6. Gráfica de actividades del 1 de abril de 2019 hasta el 1 de junio de 2019.

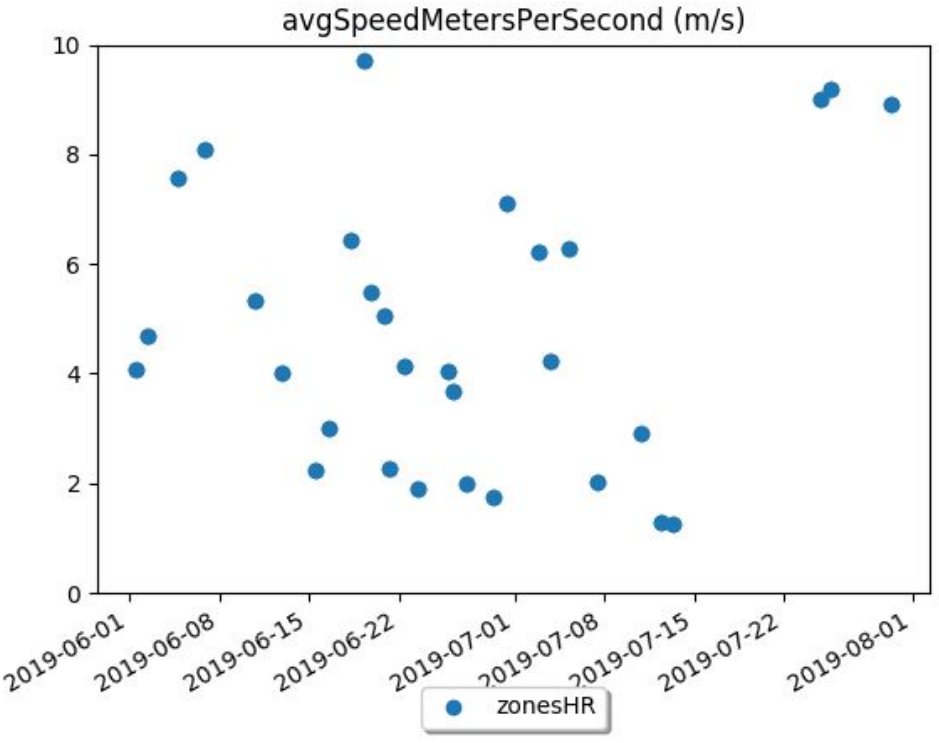


Figura 7. Gráfica de actividades del 1 de junio de 2019 hasta el 1 de agosto de 2019.

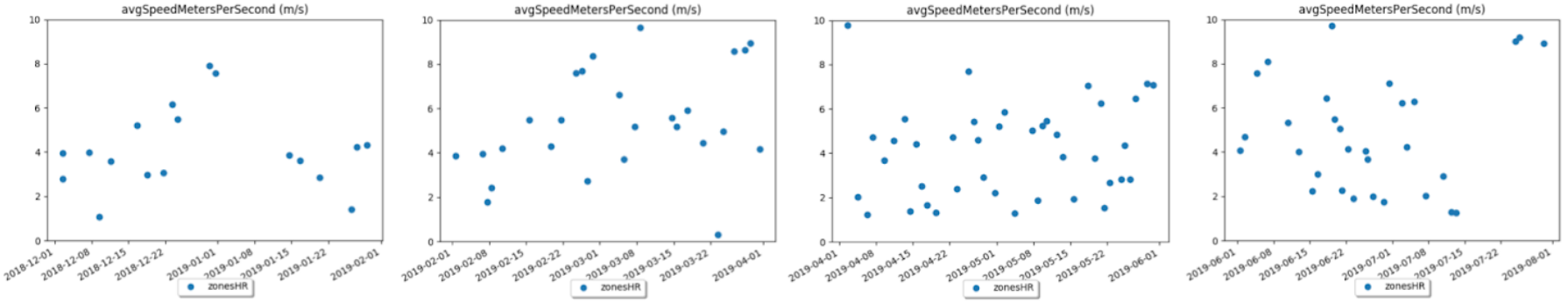


Figura 8. Gráficas conjuntas de actividades del 1 de diciembre del 2018 hasta el 1 de agosto del 2019.

Analizando minuciosamente los resultados se podría encontrar un patrón pero no es claro a simple vista. Por lo tanto, se aplica un filtro de actividades por pendiente del terreno para limpiar ruido.

Como se ha explicado anteriormente, cada parámetro calculado de velocidad, resistencia y potencia está asociado a un intervalo de esfuerzo dentro de la actividad. Este intervalo es seleccionado entre varios candidatos y consiste en un tramo del recorrido. Este tramo es posible situarlo porque consiste en un conjunto de coordenadas y además se dispone de la altitud o elevación en multitud de casos. Con estos datos es posible deducir la pendiente media del tramo en %.

En términos de ciclismo, *pendiente* se refiere a la simple inclinación de un tramo de la carretera o camino. Una carretera totalmente llana o una pista de atletismo se dice que tiene una pendiente del 0% y una carretera hacia abajo tiene una pendiente negativa.

Por ejemplo, un tramo de 5 km de desnivel positivo donde se han ascendido unos 350 metros verticales totales se dice que tiene una pendiente media de (350/5000) x 100 = **7%**.

Una pendiente de 1-2% es un poco inclinada pero nunca llega a ser costosa, se parece a tener algo de viento de frente. Del 2% al 8% hay tramos que se pueden manejar pero pueden causar fatiga con periodos muy largos y otros pueden ser incómodos para ciclistas más experimentados. Por encima del 8% se incluye un grupo de pendientes muy difíciles de aguantar en cualquier período de tiempo.

En el análisis estadístico incluido en el Anexo 2, se extrae una observación que consiste en que casi el 75% de las actividades de todos los atletas tienen una pendiente de entre el 2% y el 8%. Si filtramos las anteriores gráficas del atleta con identificador número 292 con este criterio para eliminar las actividades con más de 8% y menos de 2% de pendiente se obtiene lo siguiente.

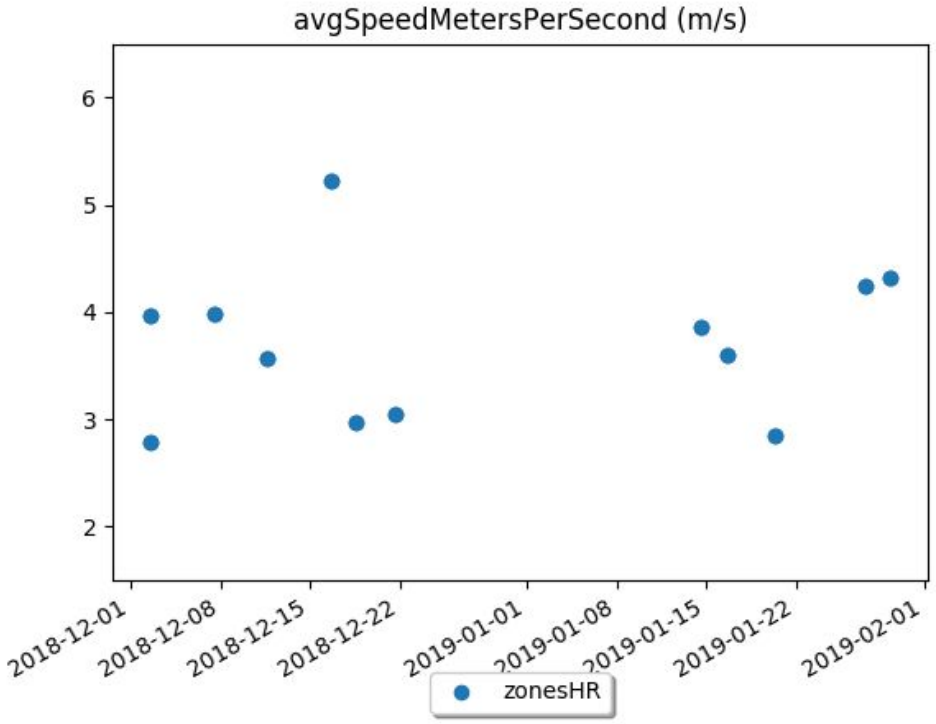


Figura 9. Gráfica de actividades del 1 de diciembre de 2018 hasta el 1 de febrero de 2019 filtradas por pendiente.

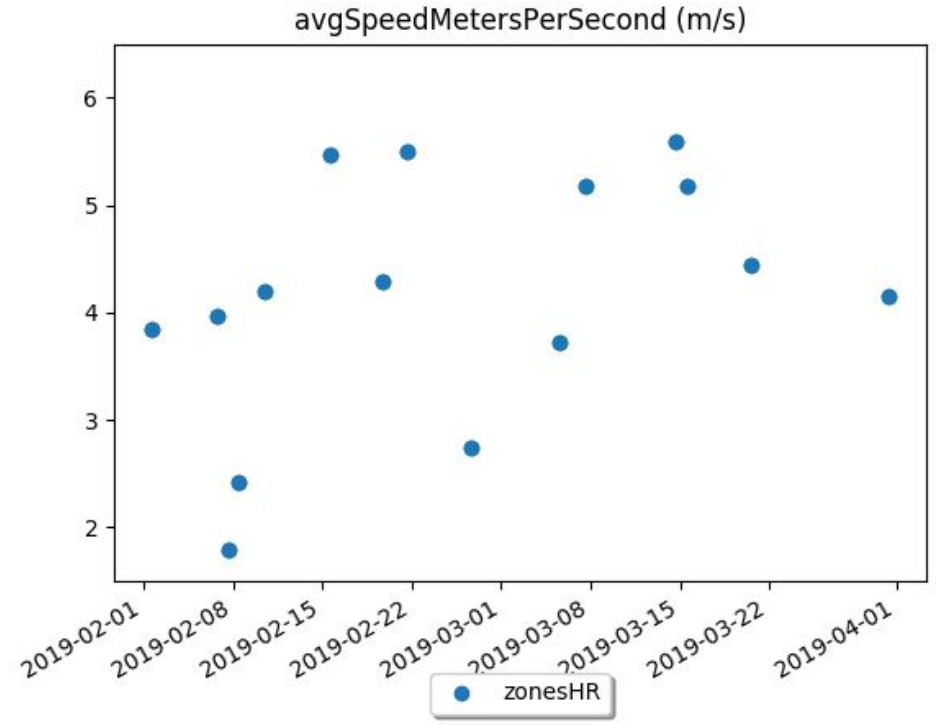


Figura 10. Gráfica de actividades del 1 de febrero de 2019 hasta el 1 de abril de 2019 filtradas por pendiente.

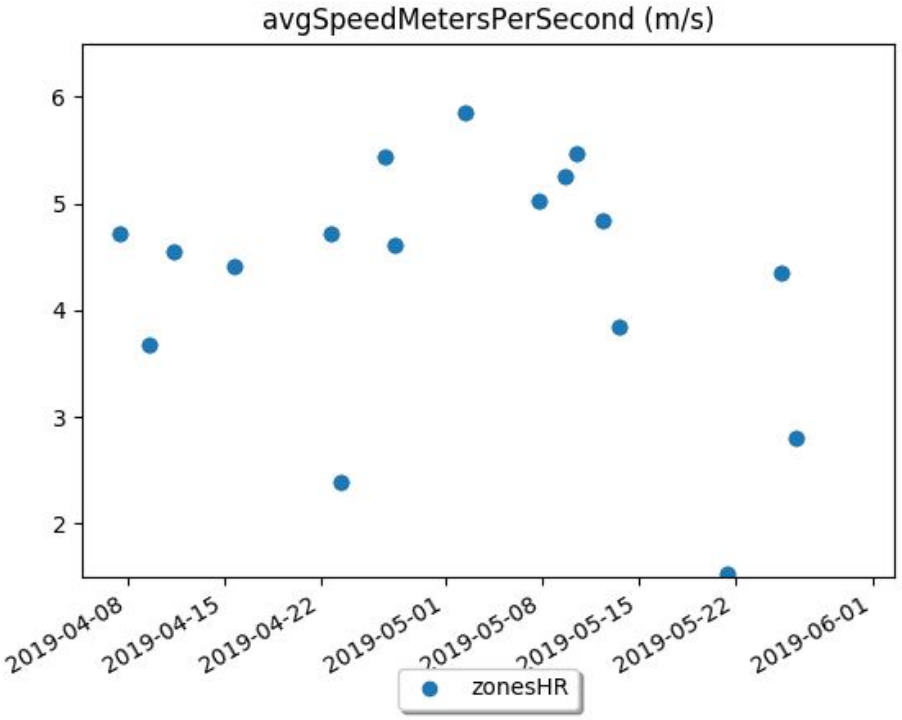


Figura 11. Gráfica de actividades del 1 de abril de 2019 hasta el 1 de junio de 2019 filtradas por pendiente.

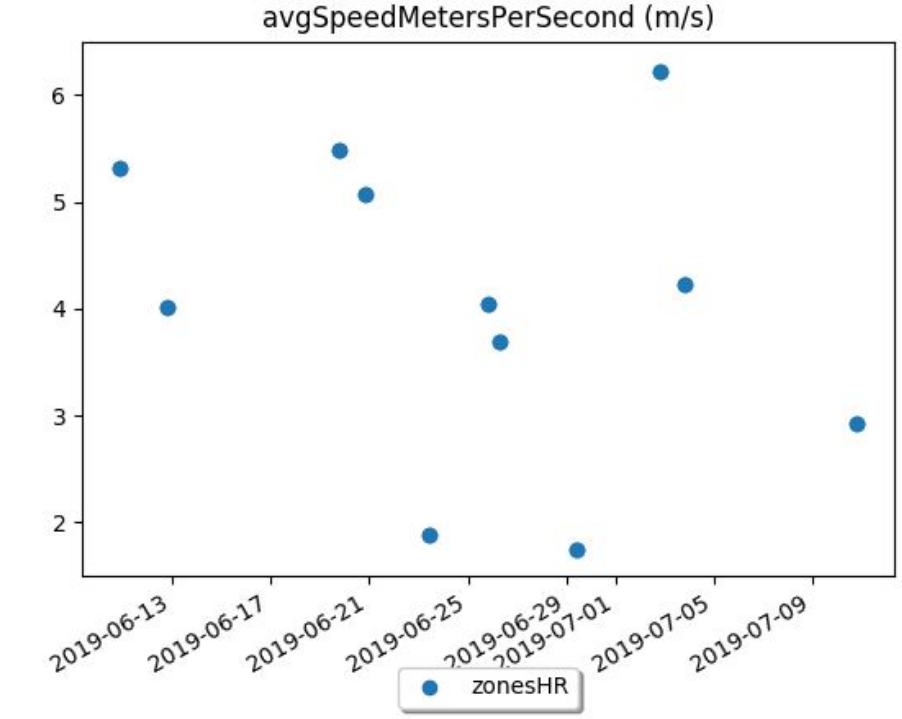


Figura 12. Gráfica de actividades del 1 de junio de 2019 hasta el 1 de agosto de 2019 filtradas por pendiente.

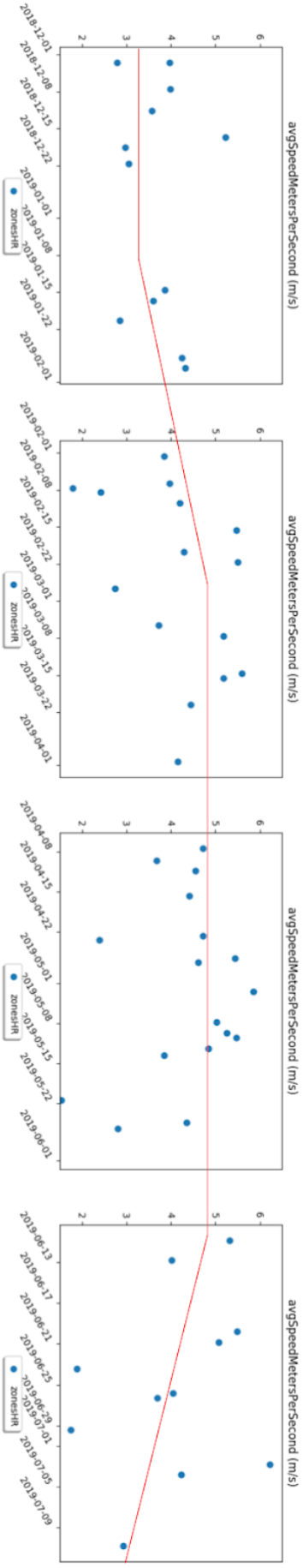


Figura 13. Gráficas de actividades del 1 de diciembre de 2018 hasta el 1 de agosto de 2019 filtradas por pendiente y con la tendencia marcada con la línea roja

En estas imágenes podemos observar que el parámetro velocidad (*avgSpeedMetersPerSecond*) sigue tres etapas bien diferenciadas: evolución constante, evolución con aumento lineal y evolución con disminución lineal.

Estas etapas siempre siguen una periodicidad y un orden. En el caso que se muestra, cuando los parámetros de velocidad están en valores pequeños se produce una evolución constante que no tiene una longitud definida. A continuación, se produce una mejora en la velocidad siguiendo una función lineal cuya pendiente es positiva y puede variar. Cuando se está llegando a los mejores registros del ciclista de nuevo se produce otra evolución constante. Seguidamente se produce una disminución en la velocidad siguiendo una función lineal con pendiente negativa. Finalmente, cuando se vuelve a llegar a los valores mínimos del ciclista, se repite el ciclo con otra evolución constante.

Por otro lado, se puede observar a simple vista que sigue permaneciendo ruido en los datos y es esencial hacer un filtrado más elaborado para que no genere ningún problema. En los siguientes apartados se explicará como se ha llevado a la práctica.

Cabe destacar que el filtrado por pendiente ha conseguido eliminar el ruido suficiente para poder descubrir un patrón. Además se puede determinar que la mayoría de datos de este ciclista en concreto oscilarán entre un mínimo de 3 y un máximo 5 con 0.5 m/s de margen.

Explorando más atletas se han encontrado resultados muy parecidos al anterior con lo que se puede afirmar que el patrón se repite y es común. Este paso es muy importante para construir un modelo predictivo y puede simplificar mucho el trabajo ya que definiendo una función lineal por partes, una por cada tramo del patrón, se conseguiría ajustar muchos datos. No obstante, podría darse el caso de observar que un conjunto de atletas siguiera otro patrón diferente. Entonces sería necesario establecer una clasificación del atleta antes de realizar la predicción y alargaría un poco más el proceso pero sería mucho más efectivo.

En definitiva, las observaciones a través de gráficas que se han realizado junto al análisis estadístico concluyen que hay un único patrón general que siguen todos los ciclistas para el parámetro velocidad del deporte de ciclismo en carretera y en el que se basarán los modelos predictivos construidos en el proyecto.

8. Casos de uso

Un caso de uso consiste en unos determinados pasos entre un actor y un sistema para conseguir un objetivo en concreto. En este proyecto se diferencian tres actores diferentes: deportista, entrenador y desarrollador. El sistema siempre será el aplicativo.

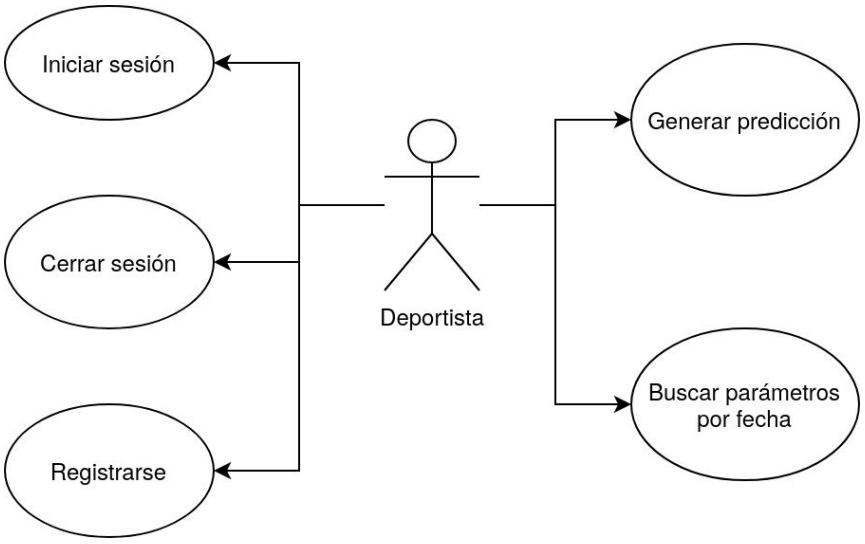


Figura 14. Diagrama de casos de uso del deportista.

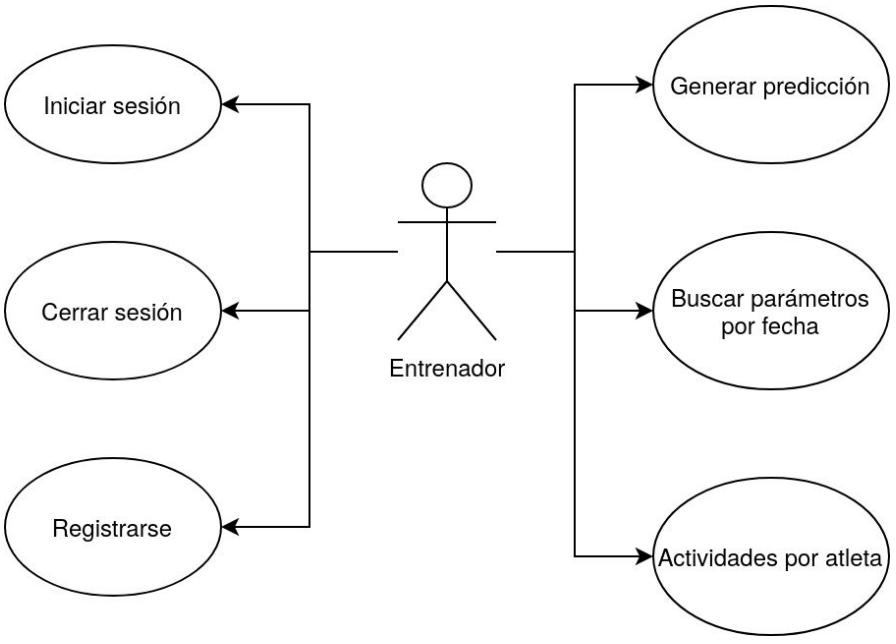


Figura 15. Diagrama de casos de uso del entrenador.

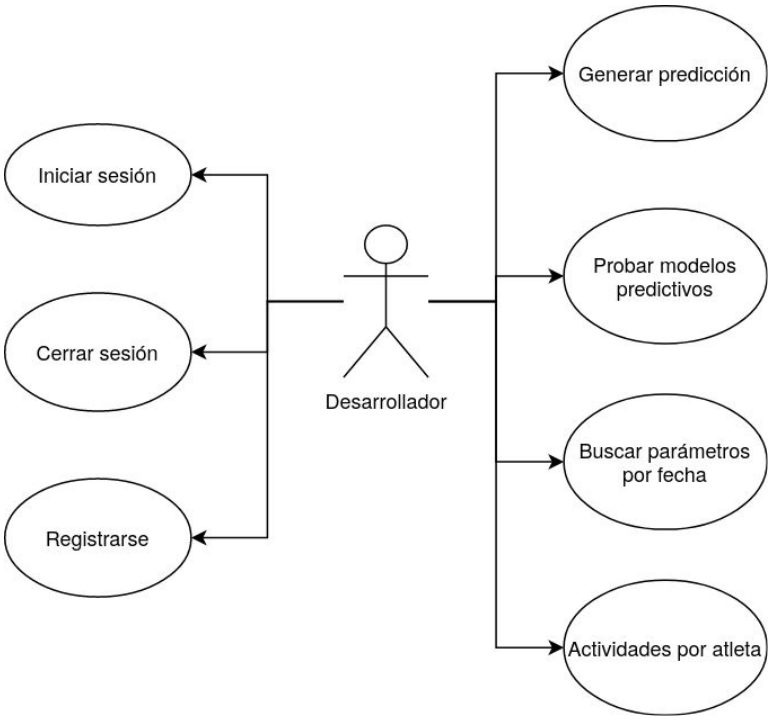


Figura 16. Diagrama de casos de uso del desarrollador.

A continuación se muestra un conjunto de tablas con explicaciones más detalladas de cada caso de uso. Contienen los siguientes apartados:

- **Actor**: La entidad que quiere ejecutar el caso de uso.
- **Disparador**: El suceso que ocasiona la ejecución.
- **Escenario principal de éxito**: El flujo que sigue el caso de uso sin ningún error.
- **Extensiones**: Alternativas del caso de éxito.

Como observación, en los casos de uso comunes de los tres actores diferentes se refiere al actor como *usuario*.

Por otro lado, la implementación de la funcionalidad de récords que se indica en la planificación ha desaparecido debido a las modificaciones en el proyecto y no tiene ningún caso de uso asociado.

Caso de uso	**Iniciar sesión**
Actor	Usuario con acceso en el aplicativo
Disparador	El usuario quiere acceder en el aplicativo
Precondiciones	El usuario dispone de las credenciales para acceder
Escenario principal de éxito	1. El sistema muestra la pantalla de inicio con pequeño formulario para introducir usuario y contraseña 2. El usuario presiona el botón de iniciar sesión 3. El sistema reconduce al usuario a la página de la funcionalidad *Generar predicción*
Extensiones	1. El usuario no introduce la contraseña correcta o quiere volver a la página de inicio a. El usuario le da a un botón para volver hacia atrás en el navegador b. El sistema comunica al usuario que su nombre de usuario o contraseña son incorrectos

Tabla 10. Caso de uso *Iniciar sesión*.

Caso de uso	**Cerrar sesión**
Actor	Usuario con acceso en el aplicativo
Disparador	El usuario quiere cerrar la sesión
Precondiciones	El usuario tiene la sesión iniciada
Escenario principal de éxito	1. El usuario selecciona la opción de cerrar sesión en el menú de navegación lateral. 2. El sistema redirige a la página de inicio.

Tabla 11. Caso de uso *cerrar sesión*.

Caso de uso	**Generar predicción**
Actor	Usuario con acceso en el aplicativo
Disparador	El usuario quiere consultar y generar predicciones de un atleta
Precondiciones	El usuario tiene la sesión iniciada y está en la pantalla de la funcionalidad
Escenario principal de éxito	1. El usuario rellena el formulario indicando el deporte, parámetro, zonas, número identificador de atleta y fechas 2. El usuario presiona el botón de generar predicción 3. El sistema muestra en forma de gráfica el resultado de la predicción junto a los parámetros que se han utilizado para testar y entrenar
Extensiones	1. El sistema no puede generar la predicción a. El sistema muestra un texto comunicando un error y recomienda cambiar los parámetros de generación

Tabla 12. Caso de uso *Generar predicción*.

Caso de uso	**Probar modelos predictivos**
Actor	Desarrollador con acceso en el aplicativo
Disparador	El desarrollador quiere probar los modelos predictivos seleccionables
Precondiciones	El desarrollador tiene la sesión iniciada y está en la pantalla de la funcionalidad
Escenario principal de éxito	1. El desarrollador rellena el formulario indicando el deporte, parámetro, zonas y número de identificador de atleta 2. El desarrollador presiona el botón de ejecutar las pruebas 3. El sistema muestra en forma de *boxplots* el rendimiento de cada modelo a modo de comparación
Extensiones	1. El sistema no puede generar las pruebas a. El sistema muestra un texto comunicando un error por falta de datos y recomienda cambiar algún parámetro de búsqueda

Tabla 13. Caso de uso *Probar modelos predictivos*.

Caso de uso	**Buscar parámetros por fecha**
Actor	Usuario con acceso en el aplicativo
Disparador	El usuario quiere hacer una búsqueda de parámetros de un atleta en unas fechas concretas
Precondiciones	El usuario tiene la sesión iniciada y está en la pantalla de la funcionalidad
Escenario principal de éxito	1. El usuario rellena el formulario indicando el deporte, parámetro, zonas, número de identificador de atleta y fechas 2. El usuario presiona el botón de buscar parámetros 3. El sistema muestra en forma de gráfica los parámetros que encuentra con las unidades adecuadas
Extensiones	1. El sistema no puede generar la gráfica a. El sistema muestra un texto recomendando cambiar algún parámetro de búsqueda

Tabla 14. Caso de uso *Buscar parámetros por fecha*.

Caso de uso	**Actividades por atleta**
Actor	Usuario con acceso en el aplicativo
Disparador	El usuario quiere consultar cuantas actividades hay en la base de datos y cuantas tiene registradas cada atleta
Precondiciones	El usuario tiene la sesión iniciada y está en la pantalla de la funcionalidad
Escenario principal de éxito	1. El usuario rellena el formulario indicando el deporte y opcionalmente fechas o número identificador de atleta 2. El usuario presiona el botón que ejecuta la funcionalidad 3. El sistema muestra en forma de tabla los resultados que ha encontrado con los atletas ordenados de mayor a menor en número de actividades
Extensiones	1. El sistema no puede generar la tabla debido a falta de datos a. El sistema no muestra ningún resultado

Tabla 15. Caso de uso *Actividades por atleta*.

9. Desarrollo

9.1 Lectura de actividades

Primero de todo hay que programar funciones que lean la base de datos. Para ello se utiliza un patrón de programación basado en repositorios. Para el ODM solo se necesitará un repositorio para la actividad ya que lleva anidado en su anterior las demás clases.

En este fichero que tiene el rol de repositorio hay una función programada que se llama *getActivitiesByDates* que tendrá como parámetros la fecha inicial y final de búsqueda, el identificador de atleta y el de deporte. Esta función realizará la consulta adecuada sobre MongoDB con un previo establecimiento de conexión.

El resultado de la consulta contiene clases de *Activity* gracias a la implementación del ODM y puede ser tratado y accedido como un objeto. No obstante, en base de datos las zonas tienen una estructura diferente y están divididas en dos documentos: *zonesHR* y *zonesPOWER*, cada una por cada tipo de zona. Es por eso que antes de retornarlo se ejecutará una función que creará la clase *Zones* con el contenido importante de los documentos a modo de preproceso.

Esta función de conversión consiste en lo siguiente, para cada zona de tipo UA definida en *zonesHR* y *zonesPOWER* primero se comprobará que se haya obtenido el tiempo mínimo en esta zona para poder aceptar los parámetros que se han registrado. En caso positivo se creará una instancia de *Zone* que se concatena a una lista vacía o a los demás casos positivos para poder expresar el resultado en una instancia de *ZoneCollection*. Por otro lado, si no se acepta o es de una zona de otro tipo se descarta su información y no se incluye en la actividad resultante.

Por último, una vez creada y asociada la *ZoneCollection* se retornan las actividades en una clase llamada *ActivityCollection* cuya función es hacer de contenedor(lista).

9.2 Modelos de predicción

Antes de explicar los pasos de implementación que se han utilizado, es importante conocer en detalle los modelos usados para realizar predicciones sobre los datos.

En total se han decidido usar tres modelos diferentes de aprendizaje supervisado.

A continuación se explicarán detalladamente los conceptos que subyacen en cada uno de los modelos, aunque a nivel de implementación se ha realizado con la librería Scikit-Learn. Asimismo, todas la gráficas generadas como resultados se han construido con ayuda de la librería de Python Matplotlib.

9.2.1 Regresión Lineal

Es el modelo más simple, con menos pasos de todos y un caso particular de aprendizaje supervisado. Aprovechando el patrón detectado en el análisis de datos, se podría construir un modelo que ajustase una recta a unos puntos en concreto y de ahí predecir valores no muy lejanos temporalmente que se ajustarán con bastante precisión a los reales.

La regresión lineal es una de las más importantes y ampliamente usadas técnicas de regresión [24]. Una de sus principales ventajas es la fácil interpretación de sus resultados.

De manera formal sería lo siguiente, la variable dependiente es el parámetro a predecir y en un conjunto de variables independientes $\mathbf{x} = (x_1, \ldots, x_r)$ que es el tiempo interpretado como fechas y donde r es el número de predictores. Se asume una relación lineal entre y y $\mathbf{x}$:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$$

Esta es la ecuación de regresión. $\beta_0, \beta_1, \ldots, \beta_r$ son los coeficientes de regresión y ε es el error aleatorio.

La regresión lineal calcula los estimadores de los coeficientes de regresión o simplemente los pesos predecidos, denotados con $b_0, b_1, \ldots, b_r$. Que a su vez definen la función estimada de regresión:

$$f(\mathbf{x}) = b_0 + b_1 x_1 + \cdots + b_r x_r$$

Esta función debería capturar las dependencias entre entradas y salidas suficientemente bien.

La respuesta estimada o predicha, $f(\mathbf{x}_i)$, para cada observación $i = 1, 2, \ldots, n$, debería ser lo más cercana posible a la correspondiente respuesta actual y_i. Las diferencias generadas de la resta $y_i - f(\mathbf{x}_i)$ para todas las observaciones $i = 1, 2, \ldots, n$, son llamados residuos. El objetivo de la regresión es determinar los mejores pesos predecidos, que son los pesos correspondientes a los residuales más pequeños.

Para conseguir los mejores pesos normalmente se minimiza la suma de los residuales cuadrados (*sum of squared residuals* o SSR, en inglés) para todas las observaciones $i = 1, 2, \ldots, n$:

$$\text{SSR} = \Sigma_i \left(y_i - f(\mathbf{x}_i) \right)^2$$

Esta será la aproximación que se implementará y tiene el nombre del método de los mínimos cuadrados ordinarios (ordinary least squares). Particularmente, se implementará

una regresión lineal simple o de una variable que es el caso más simple de regresión lineal con una sola variable independiente, $\mathbf{x} = x$.

Esta figura representa una regresión linear simple:

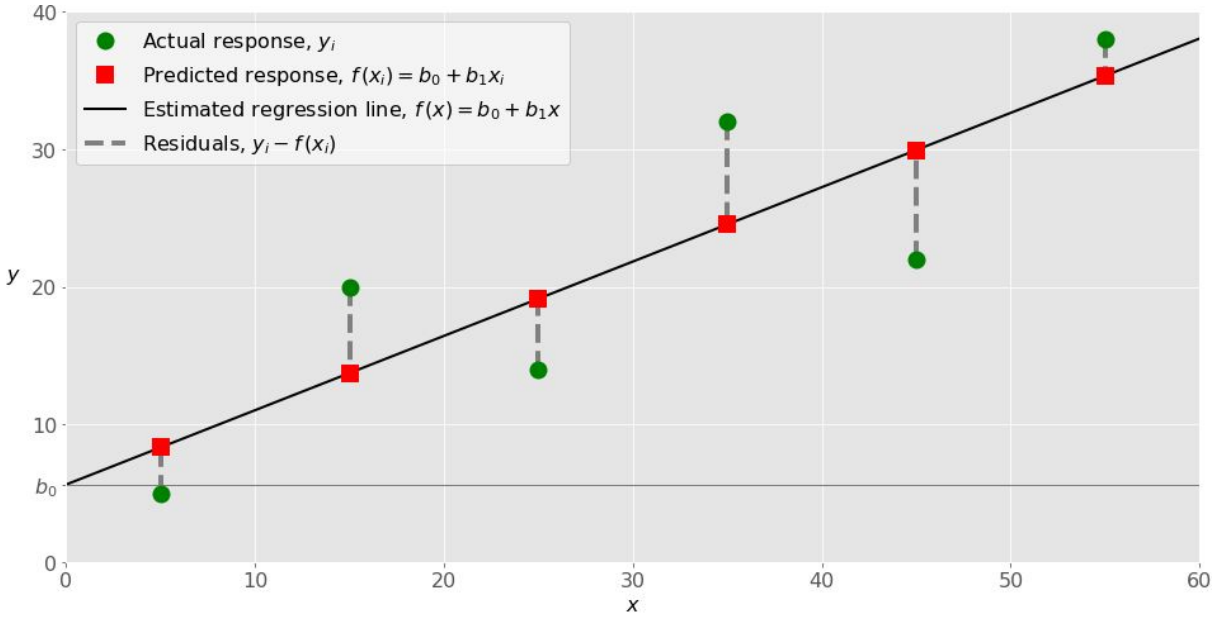


Figura 17. Gráfica de una regresión lineal aplicada sobre un conjunto de datos y_i.

Primero de todo, se dispone de un conjunto de observaciones en pares entrada-salida (x-y, en la imagen como los puntos verdes).

La función de regresión estimada (línea negra) tiene la ecuación $f(x) = b_0 + b_1 x$. Como se ha mencionado, el objetivo es calcular los valores óptimos para los pesos predecidos b_0 y b_1 que minimicen SSR y determinar la función de regresión estimada. El valor de b_0 es llamado interceptado, muestra el punto donde la regresión lineal estimada cruza la y del eje. Su valor es de respuesta estimada $f(x)$ para x = 0. El valor de b_1 determina la pendiente de la regresión lineal estimada.

Se puede observar que los residuales (líneas verticales grises) pueden ser calculados como $y_i - f(\mathbf{x}_i) = y_i - b_0 - b_1 x_i$ para $i = 1, 2, \ldots, n$. En la imagen están las distancias entre los círculos verdes y los cuadrados rojos. El método utilizado intentará minimizar esta distancia para hacer que sea lo más corta posible.

9.2.2 K-Nearest Neighbour

K-Nearest Neighbour es un algoritmo de aprendizaje supervisado que asume que el valor de un punto está determinado por los puntos que le rodean [25]. La idea principal es que la predicción consistirá en la media de los k puntos más cercanos. La distancia entre puntos en la función que se usará estará basada en la distancia Euclídea:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

El algoritmo tiene este nombre porque clasifica cada nuevo ejemplo calculando la distancia con todos los del conjunto de train. Por ejemplo, con un valor k=1, el valor será el del ejemplo más cercano del conjunto.

El principal inconveniente de este modelo es encontrar el valor de k con el que se obtenga un mayor rendimiento al clasificar [26], la técnica más utilizada es *cross validation*. Esta técnica consiste en dividir el conjunto de entrenamiento de los datos en distintos trozos, por ejemplo en 5 y clasificar cada una de esas partes mediante kNN con los valores de k que se elijan, utilizando las otras 4 partes del conjunto de entrenamiento. De esta forma se tendrá el rendimiento de kNN con todos los valores de k para cada una de las 5 particiones que se han hecho del conjunto de entreno original. Realizando una media de esos rendimientos se obtendría el valor de k más óptimo y se utilizaría para clasificar cualquier nuevo ejemplo.

No obstante, hay otra técnica que no usa siempre el mismo valor de k, sino que la k viene determinada en función de distintos parámetros como por ejemplo cuáles son los vecinos que están más cerca. La función que se usará implementa esta técnica.

9.2.3 Random Forest

Random Forest es un algoritmo de aprendizaje supervisado que utiliza un método de agregación [27], es decir combina predicciones de varios algoritmos de aprendizaje automático.

Construye una multitud de árboles de decisión en tiempo de ejecución y su salida es en modo de una predicción media de árboles individuales. La técnica que usa se llama empaquetado y entrena cada árbol de decisión en una muestra de datos diferente donde el muestreo está hecho con reemplazo.

En otras palabras es un estimador que combina el resultado de varias predicciones que agrega varios árboles de decisión con algunas modificaciones:

1. El número de atributos que pueden partirse en cada nodo está limitado en algún porcentaje del total (hiperparámetro).

2. Cada árbol dibuja una muestra aleatoria del conjunto original de datos cuando se generan para añadir un elemento de aleatoriedad de forma posterior que previene *overfitting*.

Estas modificaciones ayudan a prevenir que los árboles no tengan una correlación muy alta.

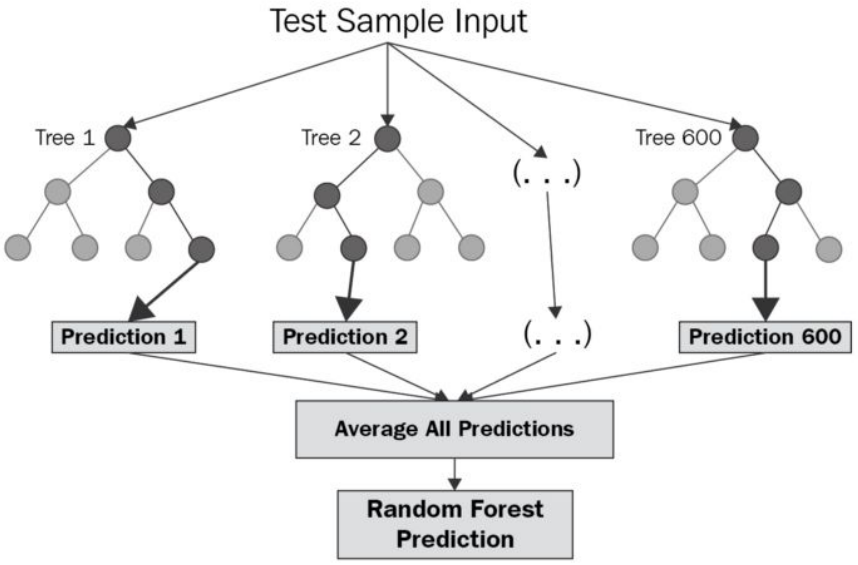


Figura 18. Estructura del algoritmo Random Forest con el detalle de la generación de árboles de predicción.

9.3 Descripción de la implementación

En este apartado, se explicará en detalle cómo se han implementado los métodos *generatePredictionLinealRegression* y *generatePredictonSupervisedLearning* que se encargan de utilizar los modelos anteriormente explicados y constituyen el núcleo que genera predicciones.

9.3.1 Preparación de actividades

Primero de todo, se realiza una preparación de los datos realizada por la función *generateGradientZonesFromActivities* haciendo una lectura de actividades de la base de datos con la función *getActivitiesByDates* (explicada en el anterior apartado) con las fechas de los parámetros de entrada. Una vez se tienen estos datos se comprueban y si no ha habido ningún resultado, ya sea porque no se han obtenido actividades o no hay zonesHR ni zonesPOWER (depende del caso), se retorna null.

En caso contrario, se produce una división de los datos en dos conjuntos: uno llamado *train* (entrenamiento) y otro *test* (prueba). La separación se produce a través de una fecha que está entre la fecha inicial y la fecha final que marca el inicio del conjunto de test. Todas las actividades que tengan fecha anterior a este parámetro de entrada son agrupadas en el conjunto de train. La función encargada de hacer esta partición se llama *generateTrainAndTestData*.

9.3.2 Filtrado de actividades

Una vez separados los datos obtenidos en la lectura, para intentar reducir el ruido excesivo que presentan se les pasa un filtro. Este filtro consiste en un valor máximo y mínimo que cambia para cada parámetro, deporte y atleta que están dados por la media y la desviación estándar de los valores.

Se produce una iteración sobre todos los datos leídos y si la diferencia absoluta de uno de los valores sobre la media es estrictamente mayor que la desviación estándar es descartado. Este método ayuda a reducir considerablemente el ruido y la cantidad de *outliers* presentes en los datos porque establece un rango muy limitado. Para obtener la media y la desviación estándar se utiliza la función *averageValuesPacked*.

Además este filtro nunca eliminará los datos en su totalidad porque las distribuciones que siguen los datos de los parámetros son unimodales. En el Anexo 2 y 3 se comprueba en un análisis estadístico detallado.

Si después del filtrado los datos no son del todo consistentes y tienen más de dos semanas sin ningún registro o son muy escasos se retorna *null* porque no serían adecuados para hacer una comparación de la predicción generada con los valores reales. Por otro lado, dependiendo de la funcionalidad que ha llamado esta función y controlado por una variable booleana, no se descarta el caso de tener datos muy escasos en el conjunto de test, es decir, hacer la predicción en una fecha futura que todavía no ha sucedido.

A partir de aquí, los datos ya han sido preprocesados para poder aplicar el modelo.

9.3.3 Regresión lineal

En el caso de regresión lineal, se obtiene la pendiente *m* del modelo. Si esta es igual o se aproxima a 0 por menos de 0.01 en valor absoluto, se construye una línea recta que consistirá en la predicción. Si los parámetros de entrenamiento son valores muy bajos que pueden hacer que la recta de regresión tome valores negativos o cercanos a cero también se opta en construir una línea recta. En cualquier otro caso se usará el modelo para generar la predicción, exactamente con la función *LinearRegression* de la librería Scikit-Learn.

En este punto, surge un problema. Las fechas de los datos de las actividades leídas no siguen una periodicidad. Al tratarse de registros de un deportista normalmente siguen una planificación establecida que va cambiando a lo largo de los meses con sus respectivos días de descanso y a más se ha de tener en cuenta de que los datos han sido filtrados y algunas sesiones no están representadas.

Para intentar solucionarlo se ha definido la variable *avgDaysBetweenActivities* que contiene una media global de los días entre las actividades de un atleta en concreto. Teniendo en cuenta que las predicciones son un conjunto de valores de salida, de una sola dimensión en la unidad del parámetro seleccionado, con la ayuda de esta variable se

genera una fecha por cada valor. El primer valor de la predicción llevará la fecha donde empieza el conjunto de test y los siguientes valores, la fecha del anterior a él más el valor de *avgDaysBetweenActivities*.

No obstante, en el caso de disponer las fechas de test se utilizarán para asignar fechas en lugar de la solución propuesta para poder realizar una mejor evaluación del resultado del modelo.

En definitiva, con estas simplificaciones se consigue una solución al problema de la no periodicidad de los registros.

9.3.4 Estructura de datos para RF y KNN

En aprendizaje supervisado el objetivo principal es aprender una función que asocia un conjunto de entrada con una salida de forma que se pueda definir un algoritmo que aprende cómo predecir parámetros de salida dados los parámetros de entrada [28]. Un ejemplo sería:

X	y
1	2
2	3
4	5
5	6
6	7
7	8
8	9

Tabla 16. Ejemplo de valores de entrada (conjunto X) y salida (conjunto y).

No obstante, el problema que se ha de resolver consiste en una secuencia de números que son valores de parámetros ordenados por un índice temporal. Por lo tanto, hay que hacer una conversión en el problema para poderlo solucionar con RF y KNN.

Para ello se usará una función incluida dentro de la clase DataFrame de la librería Pandas que se llama *shift* [29].

Esta función se utiliza para crear copias de columnas para desplazarlas hacia delante o hacia detrás. Justo lo necesario para crear columnas de observaciones con retraso temporal así como columnas para predecir observaciones para un conjunto de datos de series temporales.

Por ejemplo la siguiente serie temporal:

s
0
1
2
3
4
5
6

Tabla 17. Serie temporal *s*.

Se podría convertir en:

s	s+1
0	1.0
1	2.0
2	3.0
3	4.0
4	5.0
5	6.0
6	NaN

Tabla 18. Generación de la serie temporal *s+1* a partir de la serie *s*.

En una gráfica se podría representar como:

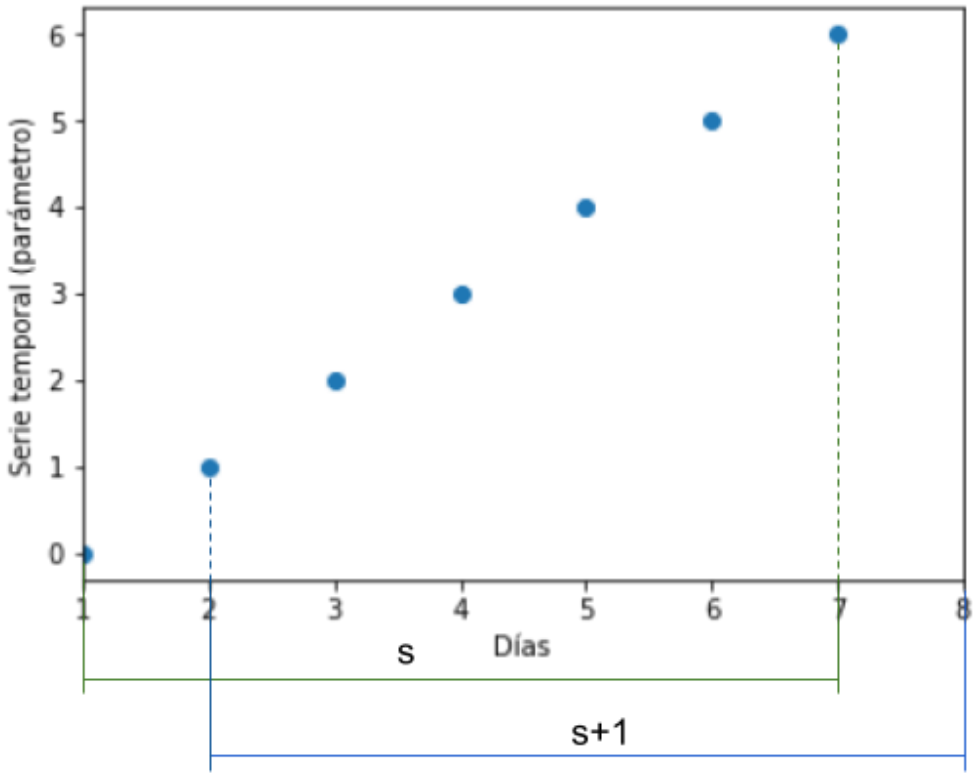


Figura 19. Gráfica representativa de las series temporales *s* y *s+1*.

En la tabla, la primera columna *s* puede ser tomada como la entrada X y la segunda *s+1* como el valor de salida *y.* De esta forma, el valor de entrada 0 se puede utilizar para predecir el valor de salida 1 (representados en la gráfica) y así sucesivamente hasta construir una estructura de datos adecuada.

Con diferentes tipos de aplicaciones de la función *shift* se puede construir secuencias de varios valores de entrada y de salida es decir para poder predecir más de un valor de salida con varios valores de entrada.

9.3.5 RF y KNN

Para la implementación de RF y KNN (función llamada *generatePredictionSupervisedLearning*) antes de aplicarse el filtrado explicado anteriormente se produce una actualización de una variable importante. Su nombre es *avgActivitiesPerMonth* y es un valor decimal que consiste en el valor medio de las actividades por cada mes del atleta de todos sus registros. Para ayudar a la actualización se utiliza una variable auxiliar llamada *allCount* que es una lista que va guardando el número total de actividades de cada mes.

Además, después del filtrado se añade la siguiente condición extra para eliminar ruido y posibles casos que no servirían para testear el modelo. Si la cantidad de datos de test es estrictamente menor que *avgActivitesPerMonth* se retorna null. Esta condición es más fuerte que las anteriores explicadas y obliga a tener una cantidad de datos abundante para así reducir casos malos para realizar pruebas. En el caso de que la funcionalidad de generar predicción haya llamado esta función no se utiliza esta condición de filtrado.

El siguiente paso es la aplicación del modelo [30] y tiene diferentes partes dependiendo del tipo. Con una variable booleana llamada RF se elige aplicar Random Forest si es cierta o K-Nearest Neighbors si es falsa. Las funciones específicas de la librería externa Scikit-Learn son *RandomForestRegressor* y *KNeighborsRegressor*. A parte, con otra variable llamada NtoN se escoge aplicar una predicción con entrada N y salida N o entrada N y salida 1 repetida N veces como se ha explicado anteriormente.

En ambos casos, se aplica la función llamada *series_to_supervised* explicada con detalle en el apartado anterior que transforma las series temporales en el formato necesario para poder aplicar regresión con KNN o RF.

Una vez los datos han sido transformados, la N finalmente coge el valor redondeado de *avgActivitiesPerMonth*. Este valor es escogido para solucionar el problema generado de la variabilidad del número de registros del deportista a lo largo del tiempo, justo como en regresión lineal pero sobre un mes. De esta forma se realiza una aproximación en forma de número entero que consiste en la media de actividades que se registran por mes de un deportista y normaliza la variabilidad proporcionada por los datos.

A diferencia de regresión lineal, ahora se utilizan todos los datos de registros del atleta para entrenar el modelo y se introducen los últimos N valores para ejecutar la predicción. En el caso de *Nto1* se realiza esta operación N veces añadiendo a cada iteración la salida del modelo a la entrada hasta conseguir la salida deseada. Con un modelo u otro, finalmente se consigue un conjunto de N valores que consiste en la predicción.

En este punto, durante el desarrollo del proyecto se encontró un problema. Aún habiendo eliminado ruido con los filtros establecidos, la salida de aprendizaje supervisado son un conjunto de valores exactos que seguirán presentando ruido aunque sea en un grado menor. Por lo tanto, es bastante difícil que este conjunto de valores se acerque con

mucha exactitud a los datos de test así que se procedió a resolver este problema con la siguiente solución.

Para poder normalizar los datos generados, se aplicará regresión lineal para conseguir una recta ajustada a estos puntos. Esta recta conseguirá suavizar el posible ruido presente en los valores de salida. La función utilizada será *executeLinealRegressionSL* que tendrá como entrada la predicción generada en el apartado anterior. Esta función convierte los valores de la predicción y genera fechas en dos arrays diferentes en el formato adecuado para ejecutar la función *fit* del modelo.

La salida de aprendizaje supervisado también genera unos valores que no tienen ninguna fecha asociada, por lo tanto para poder aplicar regresión lineal es necesario simularlas. Para ello se utiliza la variable *avgDaysBetweenActivities* como se ha explicado anteriormente.

Para acabar, la salida de la función de regresión lineal consiste en el resultado de la predicción y se envía a las funciones que generan las gráficas.

En resumen, los métodos de predicción que se utilizan en este proyecto se pueden englobar en tres: **regresión lineal simple**, **K-Nearest Neighbors más regresión lineal** y **Random Forest más regresión lineal**.

9.3.6 Evaluación de la predicción

Por último, una vez se ha aplicado el modelo correspondiente se obtiene un conjunto de valores con sus respectivas fechas que es la respuesta al problema planteado y se puede mostrar en una gráfica.

En el caso de la funcionalidad de test de modelos, es necesario obtener una medida del error de la predicción. Para ello se realiza una comparativa con los datos del conjunto de test.

El método estándar para datos cuantitativos es RMSE (*Root Mean Square Error*) [31]. Que se define formalmente como:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

- $\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n$ son los valores de la predicción
- $y_1, y_2, \ldots, y_n$ son los valores de test
- n es el número total de observaciones

Cabe destacar que un RMSE igual 0 significa que los valores de la predicción son exactamente iguales a los valores de test y por lo tanto la predicción sería perfecta.

Finalmente, se comprueba si la opción de generar una gráfica es cierta y si es así se llama a la función *plotZonesParametersPrediction* que se encarga de generarla. Tanto la función *generatePredictionLinealRegression* como *generatePredictionSupervisedLearning* retornan una tupla de dos valores: el RMSE y la gráfica como una instancia de la clase Matplotlib que proviene de la librería *matplotlib* o null si la opción era falsa.

9.4 Optimización de parámetros de entrada de RF y KNN

En el proyecto, se han establecido de forma arbitraria unos parámetros de entrada de los modelos por simplificación. Por ejemplo, los datos con los que se entrenan RF y KNN son todos los registros del atleta hasta la fecha y la ventana de número de actividades con la que se ejecuta la función de predicción del modelo se ha definido con una variable arbitraria *avgActivitiesPerMonth*.

No obstante, estos valores podrían optimizarse para mejorar el tiempo de ejecución y la eficiencia de los modelos. Para ello es necesario realizar pruebas para poder observar el rendimiento de los modelos con diferentes parámetros de entrada y analizar los resultados.

Por ello se ha procedido a realizar la siguiente prueba. Primero de todo, es necesario un atleta y se ha elegido el que tiene número identificador 1504 porque es el que tiene más actividades en base de datos del deporte *CYCLING_STREET*.

En todas las actividades de este atleta, se procederá a realizar predicciones con un modelo en concreto. Se ha elegido la variante RF-Nto1, que consiste en el modelo Random Forest. El programa de la prueba realizará una iteración sobre todas las actividades del atleta con tres punteros que delimitan el conjunto de train y el conjunto de test. Concretamente un puntero para el inicio de train, otro para separar train de test y el último para marcar el final de test. En cada iteración realizará una predicción de la cual se guardará el tiempo total de ejecución en caso de generar un predicción exitosamente y el valor RMSE de la predicción.

Como se ha mencionado, el primer parámetro a optimizar son los datos con los que se entrena el modelo que se nombra de aquí en adelante como *Longitud de serie temporal* y su unidad son días. El segundo parámetro corresponde a la longitud de la ventana de predicción, se llama *N* como en el resto del proyecto y su unidad son actividades.

El programa de la prueba se ejecutará con valores de parámetros diferentes a cada vez y se recogerá el tiempo medio de ejecución en segundos y la media de RMSE de todas las predicciones realizadas en los datos del atleta.

Los resultados obtenidos son los siguientes:

Días				
365	1,365	1,973	2,569	2,992
300	2,140	1,917	3,567	2,823
200	1,483	1,645	2,159	2,576
150	1,353	1,368	1,977	2,210
120	1,289	1,339	2,111	2,296
90	0,983	1,295	1,598	2,469
60	1,145	1,206	1,754	2,043
30	0,885	1,136	-	-
N	5	7	10	13

Tabla 19. Tabla de la media de tiempo de ejecución obtenida(s). En el eje horizontal la N y en el eje vertical la longitud de la serie temporal. Los colores de las casillas van de un rango de verde a rojo, donde el más verde es el valor más pequeño y el más rojo el más grande en relación a la media.

Días				
365	0,741	0,734	0,727	0,737
300	0,745	0,742	0,731	0,736
200	0,751	0,756	0,731	0,733
150	0,754	0,748	0,746	0,751
120	0,759	0,767	0,745	0,752
90	0,780	0,834	0,820	0,875
60	0,905	0,864	0,745	0,808
30	0,857	0,986	-	-
N	5	7	10	13

Tabla 20. Tabla de la media de RMSE obtenida. En el eje horizontal la N y en el eje vertical la longitud de la serie temporal. Los colores de las casillas van de un rango de verde a rojo, donde el más verde es el valor más pequeño y el más rojo el más grande en relación a la media.

Como podemos observar, estas tablas muestran el resultado de la prueba para el tiempo de ejecución y para la media de RMSE respectivamente.

En la segunda tabla es fácil observar que hay un límite en el cual si se aumenta la longitud de la serie temporal el RMSE medio no mejora. Con ayuda de las siguientes gráficas se intenta establecer de forma numérica el valor preciso del límite:

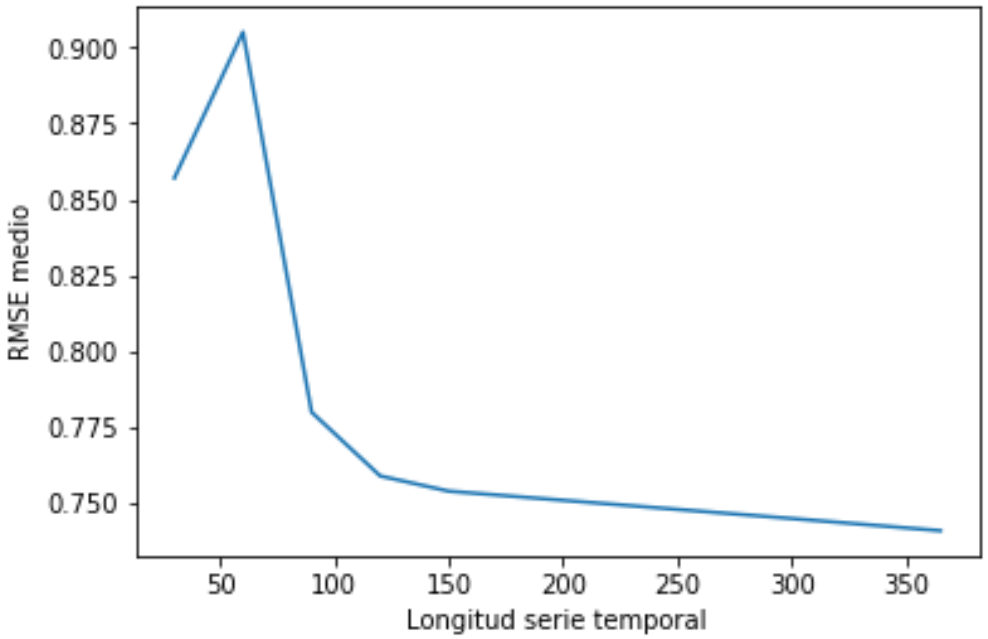


Figura 20. Gráfica del RMSE medio en relación a la longitud de la serie temporal para un tamaño de ventana de 5 actividades.

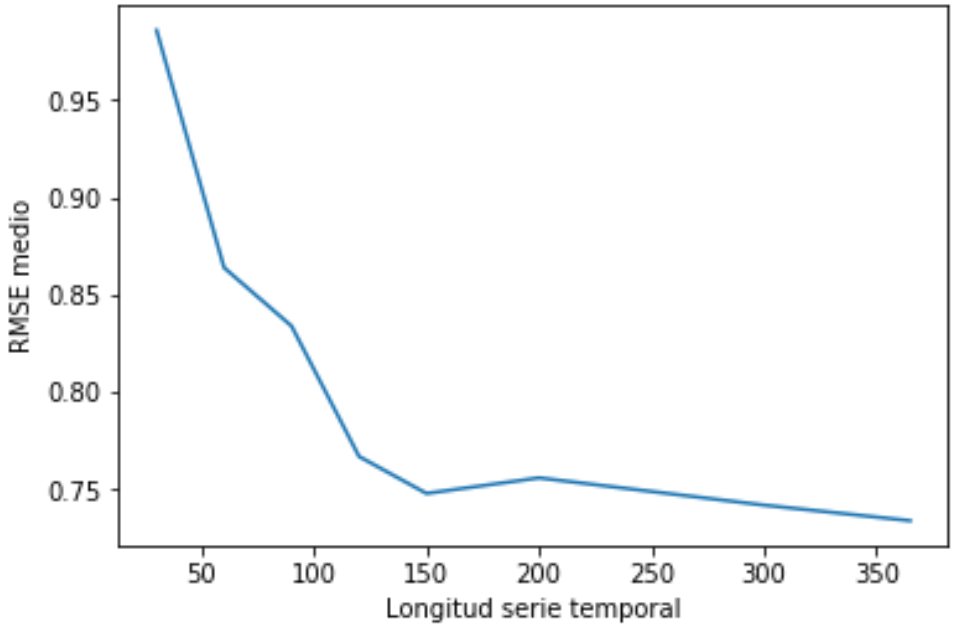


Figura 21. Gráfica del RMSE medio en relación a la longitud de la serie temporal para un tamaño de ventana de 7 actividades.

En estas gráficas se observa una mejora importante desde valores pequeños hasta 150 días. A partir de ese valor se aprecia más estabilidad en los resultados que son cercanos a 0,75. Por lo tanto se puede concluir que en un valor aproximado de 150 días se encuentra un límite de mejora. Un aumento en cantidad de datos por encima de este límite no mejorará el rendimiento del modelo.

Respecto a la tabla del tiempo de ejecución es fácil observar que cuanto más grande es la serie temporal más tiempo cuesta generar la predicción ya que la lectura de actividades es sobre una cantidad de datos mayor. Además la *N* también influye en el tiempo de ejecución ya que en este modelo se ejecutan predicciones en un bucle de forma que contra más grande sea aumenta más el tiempo de ejecución.

Por otro lado, en las tablas se puede observar que el tamaño de la ventana no interfiere con los valores RMSE aparentemente. Teniendo en cuenta todos estos factores y que en las predicciones se quiere generar como mínimo un mes, **el valor de *avgActivitiesPerMonth* definido anteriormente parece ser el óptimo** ya que es diferente para cada atleta y es justamente el número de actividades necesarias sin excederse.

Finalmente, como el tiempo de ejecución aumenta proporcionalmente a la serie temporal, **el valor de 150 días parece ser el valor adecuado** ya que como se ha concluido anteriormente es un límite y ningún otro valor superior a éste mejora sustancialmente el rendimiento del modelo.

El experimento que se ha realizado se ha centrado en un solo atleta y modelo intentando generalizar los resultados. No obstante, queda como trabajo futuro realizar experimentos en otros atletas y otros modelos para averiguar si los parámetros que se adaptan mejor a sus datos son diferentes.

9.5 Pantallas

9.5.1 Buscar parámetros por fechas

Esta funcionalidad está representada con el nombre *Search parameter zones by dates*. La pantalla principal tiene esta forma:

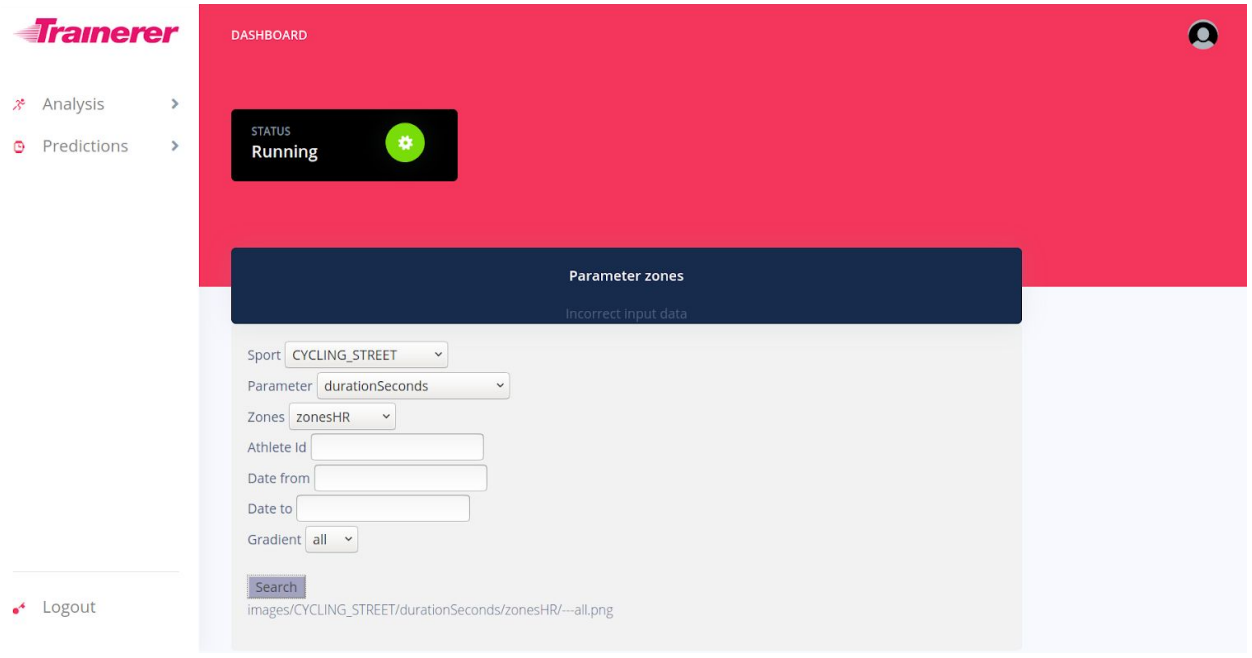


Figura 22. Pantalla de la funcionalidad *Buscar parámetros por fecha*.

En el formulario se da la opción para escoger el deporte, parámetros, zona, identificador de atleta, fecha de inicio, fecha final y pendiente de la actividad. La pendiente tiene utilidad solamente en ciclismo y *running* y sirve para filtrar los resultados. Tiene 4 opciones:

- *all*
- *0 to 2*
- *2 to 8*
- *>8*

Estas opciones están basadas en el valor de la pendiente donde se ha obtenido el parámetro correspondiente.

La tarjeta de la cabecera que indica el *status* es común en todas las funcionalidades. Indica si localmente se está ejecutando Flask en modo *debug*, si es así la tarjeta indica *Running* con el icono en verde en caso contrario muestra el mensaje *Stopped* con el icono en rojo.

Una vez se presiona el botón y si no hay ningún error, se muestra la salida en forma de gráfica.

Figura 23. Gráfica generada por la funcionalidad *Buscar parámetros por fecha*.

9.5.2 Actividades por atleta

El nombre en el menú lateral es *Activities by athlete* y automáticamente genera una tabla de ejemplo con una entrada por cada atleta con la siguiente información: identificador, número de actividades total en la base de datos, número de zonas de tipo HR dentro del rango UA o superior a lo largo del conjunto de actividades y lo mismo que lo anterior pero para número de zonas de tipo POWER.

Además, dispone de un pequeño formulario que permite seleccionar deporte, fechas y/o identificador de atleta para consultas individuales.

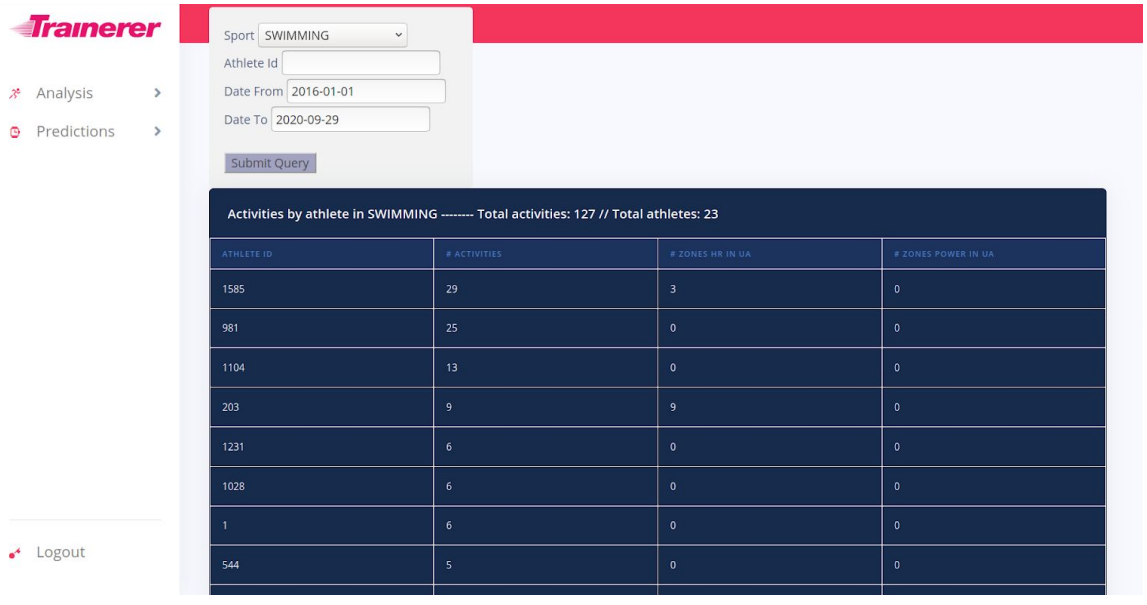


Figura 24. Pantalla de la funcionalidad *Actividades por atleta*.

En esta figura se puede observar un ejemplo de resultado de consulta con deporte SWIMMING, fecha inicial 1-1-2016 y final 12-9-2020. Los resultados vienen ordenados por orden descendente de número de actividades siendo el primero el atleta número 1585.

9.5.3 Probar modelos predictivos

Esta funcionalidad está situada en la barra de navegación y tiene como nombre *Test predictive models*. La pantalla principal es la siguiente:

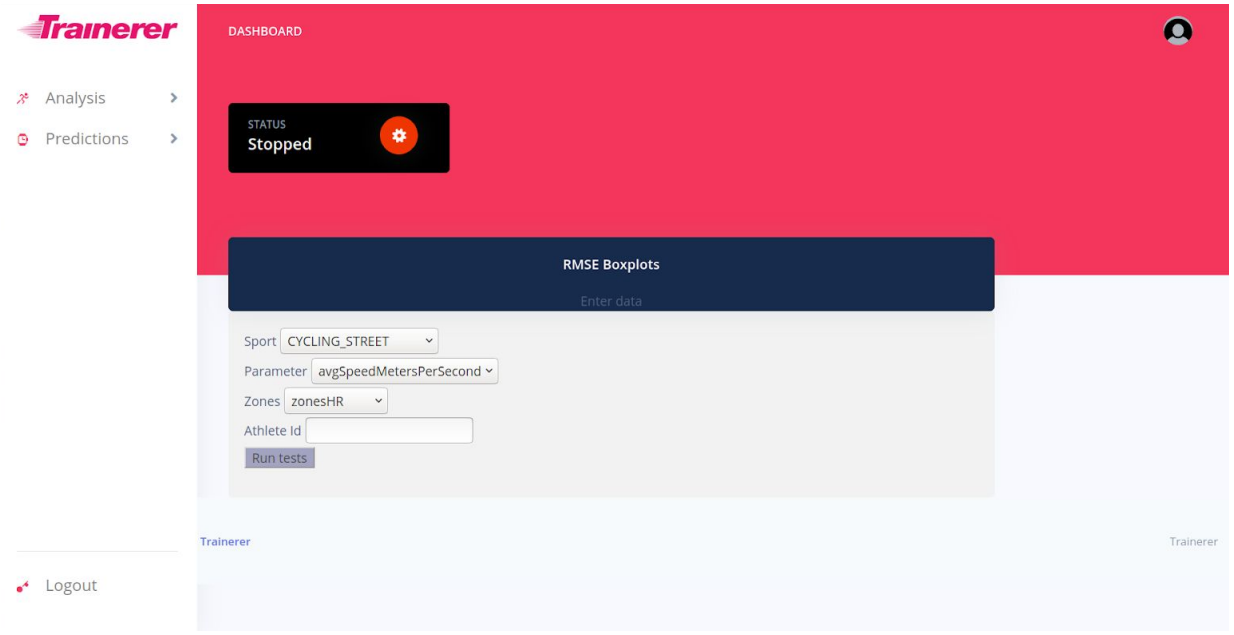
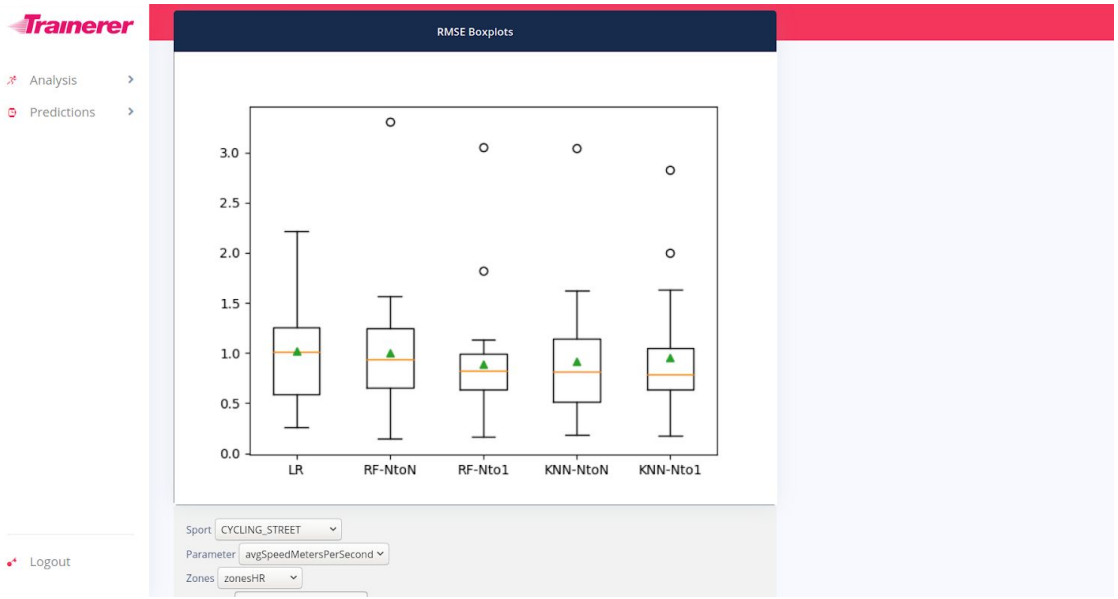


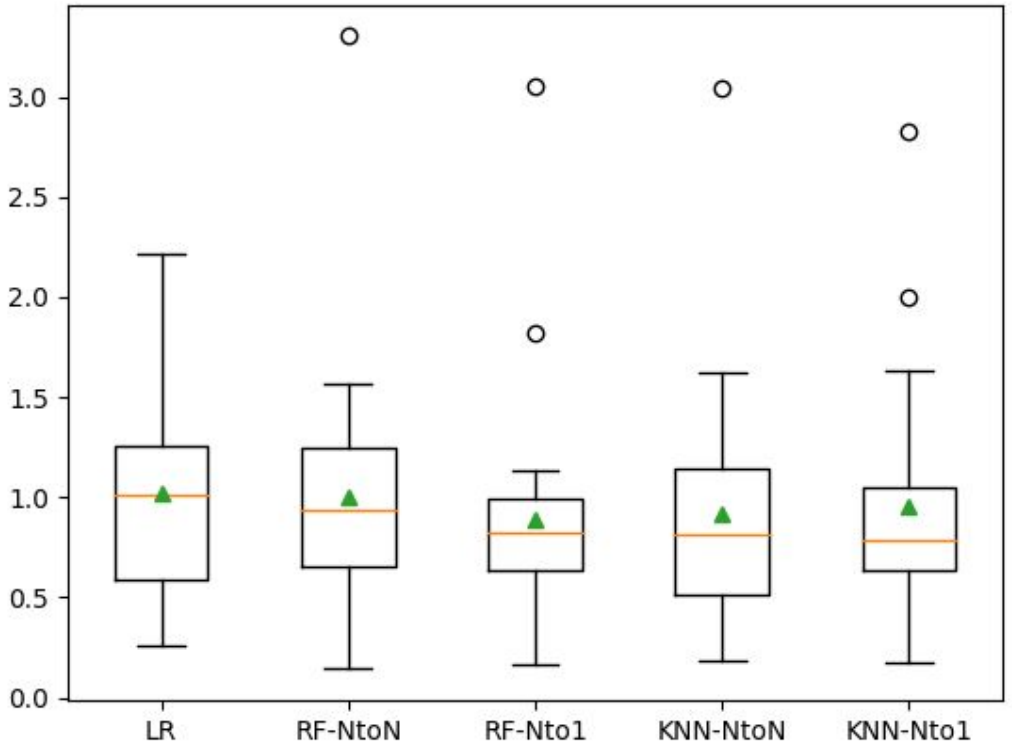
Figura 25. Pantalla de la funcionalidad *Probar modelos predictivos*.

En los desplegables da opción a seleccionar el deporte, el parámetro, el tipo de zonas y el número identificador del atleta en el que se quieren ejecutar los tests. Junto a los desplegables hay un botón llamado *Run tests* que ejecuta la funcionalidad.

La ejecución de la funcionalidad es bastante lenta porque realiza pruebas exhaustivas en todos los datos registrados del atleta con todos los modelos disponibles.

La salida que se obtiene es en forma de *boxplots*.





Figuras 26 y 27. Gráfica de *boxplots* generada por la funcionalidad *Probar modelos predictivos*.

Cada uno de estos *boxplots* corresponde a un modelo diferente. El testeo se ha realizado de la siguiente forma: para cada modelo, se realiza un bucle a lo largo de todos los registros temporales del atleta del más antiguo al más nuevo cogiendo una ventana de dos meses. Esta ventana escogida se parte en dos, la parte más antigua hace de datos de training y la más nueva de datos de test. El resultado en términos de RMSE se guarda en

una lista que recogerá todos los resultados de este atleta. Esta lista es la que está representada en el *boxplot*.

Los círculos blancos son valores con una desviación estándar elevada y probablemente *outliers*. Los triángulos verdes representan la media de todos los valores.

En el caso que muestra la foto, los *boxplots* que más se acercan al 0 son de los modelos RF-NtoN y KNN-Nto1. En otras palabras, Random Forest con predicciones de N valores de salida y K-Nearest Neighbours con generación de predicciones de 1 valor de salida.

9.5.4 Generar predicción

Esta funcionalidad se llama *Generate predictions* en la barra de navegación y tiene un formulario para poder generar una sola predicción con los siguientes campos: deporte, parámetro, zonas, el identificador del atleta, la fecha inicial, la fecha final y el modelo.

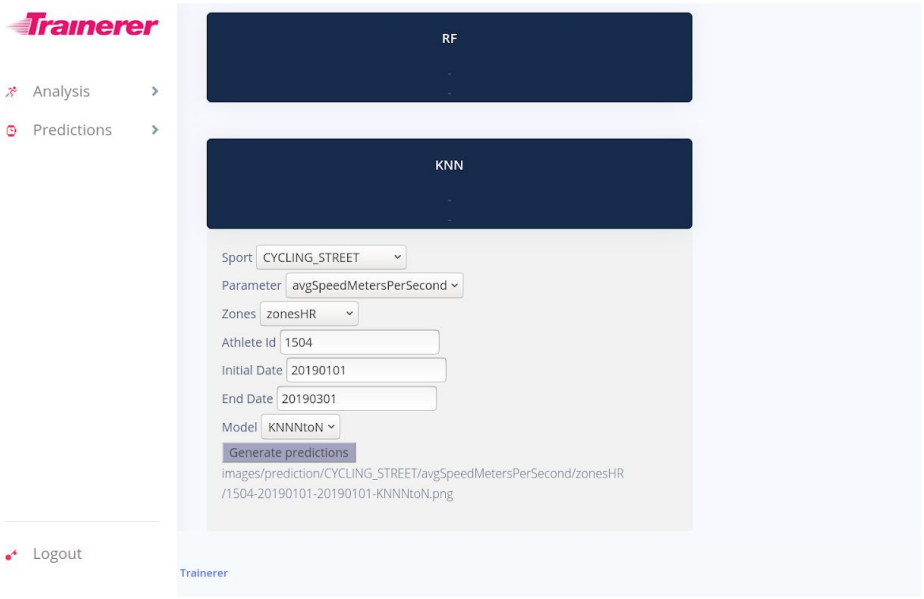
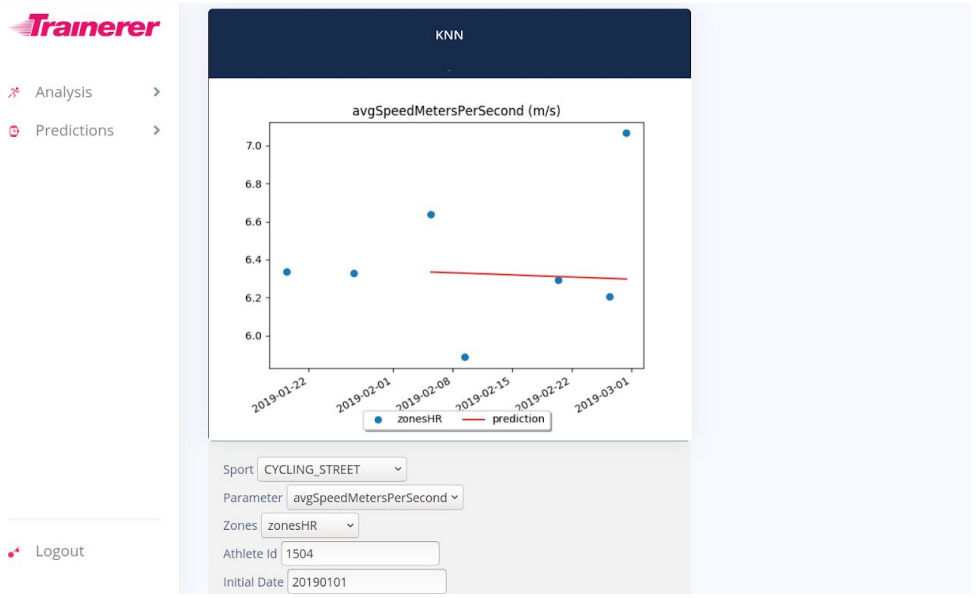
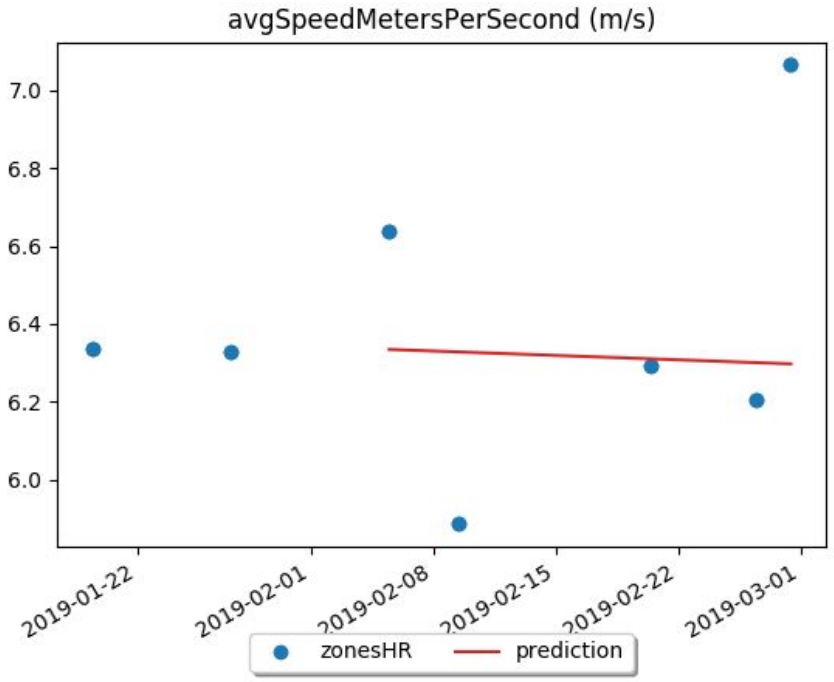


Figura 28. Pantalla de la funcionalidad *Generar predicciones*.





Figuras 29 y 30. Gráfica de la predicción generada por la funcionalidad *Generar predicciones*.

Esta gráfica es un ejemplo de salida de la funcionalidad. Los puntos azules son los valores de los registros que tiene el atleta y la línea roja consiste en la predicción generada por el modelo. En esta predicción en concreto corresponde al modelo KNN y tiene un buen resultado aunque siga permaneciendo un ruido más suave en los datos. Los puntos azules en el tramo de la línea roja no siempre se muestran y corresponden a datos de prueba para calcular la calidad de la predicción. En este caso se muestran porque la predicción está realizada en un período de tiempo pasado. Por simplificación, se asume que el usuario siempre introducirá un rango de tiempo de 2 meses, donde los datos del primer mes se utilizarán de entrenamiento del modelo y los datos del segundo mes como datos de prueba en caso de existir.

10. Sostenibilidad

En este apartado se va a explicar la repercusión económica, ambiental y social del proyecto.

10.1 Dimensión económica

El coste de realización se ha estimado de la forma más objetiva posible teniendo en cuenta los recursos humanos, hardware y software. Además se han identificado los costes indirectos y se han valorado los posibles riesgos y efectos asociados. Es importante tener en cuenta estas variaciones para no hacer una mala estimación que sea escasa ni tampoco que se exceda del precio que se acabará gastando ya que es una pieza fundamental y con la que el proyecto va a crecer y depender.

Los aspectos de costes se resuelven de la siguiente forma. Simplemente se usará lo esencial y necesario, se intentará obtener la opción más barata en todo momento como por ejemplo en lo mencionado en cuanto a software que son todo servicios gratuitos aunque algunos tienen versiones de pago. Respecto al salario está subvencionado por parte de Trainerer y de la UPC al realizarse en un convenio de prácticas.

El principal objetivo del proyecto es enfocar desde otra perspectiva los servicios ya existentes para poder darles un significado más útil y avanzado. Es por eso que realmente no se dispone de una solución alternativa que ya exista y hace que el proyecto sea bastante innovador.

A pesar de que los objetivos del proyecto son bastante ambiciosos vale la pena asumir los costes para poder realizar un avance importante en el contexto del entrenamiento personal de deportistas.

10.2 Dimensión ambiental

Primero de todo, no es trivial hacer una estimación del impacto ambiental cuando el proyecto no tiene ninguna relación con residuos materiales. Lo único que se podía tener en cuenta es el consumo eléctrico que puede llegar a generar en los diferentes servidores en los que se levantarán instancias de las funcionalidades del proyecto, pero que se puede obviar porque es relativamente muy pequeño.

Se ha planteado minimizar el impacto reutilizando recursos como el ordenador portátil cuya propiedad es del investigador y no se ha planteado comprar ninguno nuevo al igual que el ordenador de la empresa que se utiliza también en otros proyectos. Lo mismo pasa con los recursos software que han sido utilizados y en un futuro se pueden utilizar en otros proyectos y ordenadores.

Los problemas planteados en el proyecto se resuelven siguiendo la siguiente metodología: se implementa y prueba el trabajo en una máquina local para luego poderlo desplazar a un servidor remoto para poderlo hacer accesible via app y web. Por otro lado, se puede observar que la solución del proyecto proporcionará una mejora ambiental respecto a las existentes ahorrando muchos recursos en términos de consumo y electricidad, ya que no será necesario seguir visualizando aplicaciones como Strava y Garmin que proporcionan una gran cantidad de datos que tardan horas en ser analizados manualmente.

10.3 Dimensión social

A nivel personal, la realización de este proyecto proporcionará al investigador la posibilidad de estar dentro de una empresa para ver y aprender su funcionamiento; poder participar en un proyecto bastante grande en el que hay establecido un negocio actualmente y poder utilizar conocimientos sobre aprendizaje automático para poder desarrollar e iniciar una innovación en el contexto del proyecto.

Desde el punto de vista social la solución propuesta proporciona una mejora en la calidad de vida ya que va a permitir que multitud de deportistas puedan tener información de una manera más sencilla y eficaz sobre su rendimiento y como está previsto que evolucione durante el tiempo, cosa que es muy difícil de saber con las soluciones existentes.

Aunque los resultados son una aproximación sí que existe una necesidad real del proyecto y puede dar mucha información en temas de investigación sobre modelos de regresión en aplicaciones reales. Si el proyecto llegara a una fase más avanzada de la que se ha obtenido se conseguiría un gran avance en el sector pudiendo ofrecer una herramienta muy potente de la que se han realizado los primeros pasos.

11. Trabajo futuro y conclusiones

11.1 Trabajo futuro

Como se ha explicado a lo largo del trabajo, se ha decidido explorar en detalle un parámetro de un solo deporte. Ahora sería el momento de seguir analizando otros parámetros y observar los parecidos entre ellos para aplicar los mismos modelos predictivos o usar otros diferentes.

A modo de ejemplo, se ha analizado el deporte *RUNNING* detalladamente en el Anexo 3. También se ha analizado el parámetro *durationSeconds* y *avgPowerWatts* del deporte *CYCLING_STREET*.

Después de realizar el análisis, los resultados principales han sido que *avgPowerWatts* es un parámetro que sigue una distribución unimodal bastante parecida a *avgSpeedMetersPerSecond,* por lo tanto es muy probable que los modelos utilizados en este proyecto también tengan efectividad. No obstante, *durationSeconds* tiene un rango de valores mucho más grande y sus datos siguen otro tipo de distribución. Es bastante probable que los modelos implementados no acaben de funcionar bien en este parámetro y tengan un error considerable.

A parte de implementar modelos en otros deportes, la parte más importante que queda como trabajo futuro es desplegar las funcionalidades de forma efectiva dentro de la aplicación web inicial para que Trainerer pueda beneficiarse de los resultados de este proyecto en su totalidad. Una buena propuesta para dar escalabilidad al proyecto frente a este reto, sería dejar de utilizar las variables globales que se definen dentro las funciones principales de las predicciones e introducir el uso de MariaDB que finalmente no se ha acabado aprovechando. Por ejemplo se podría guardar en base de datos una tabla con una relación entre identificador de atleta y el valor de la media de sus actividades por mes. Además se podría estudiar cómo guardar los parámetros de las actividades que se extraen de MongoDB en MariaDB y si realmente producirían una mejora en tiempo de ejecución.

Por otro lado, al principio de la memoria se menciona que puede ser interesante tener comunicación con otro proyecto que genera planificaciones de entrenamiento. Si estas planificaciones fueran accesibles podrían ser de gran ayuda para generar predicciones. No es lo mismo predecir un cierto parámetro sabiendo que en el siguiente mes van a haber muchos entrenamientos donde se va a intentar rendir al máximo o bien que haya semanas de descanso y entrenamientos de recuperación. No obstante, a parte de establecer una comunicación efectiva y sin errores entre los dos proyectos, sería una tarea bastante compleja tener en cuenta todos estos factores y desarrollar un programa que decidiera utilizar un modelo u otro. Lo más ideal sería comenzar por aproximaciones pero se deja como trabajo futuro.

En el aplicativo desarrollado se da la posibilidad al atleta de examinar cinco variantes de los modelos predictivos. Si la funcionalidad estuviera desplegada dentro de una aplicación con usuarios reales, no sería muy efectivo en términos de interacción mostrar tantas gráficas. Estaría muy bien que solo se pudiera dar a elegir entre un par de modelos como máximo que el proyecto estableciera como mejores para ese atleta. Además se debería tener en cuenta el caso de no tener suficientes datos para generar una predicción y el sistema debería comunicarlo al usuario.

En definitiva, el trabajo consiste esencialmente en una aproximación hacia el desarrollo en una plataforma de unas funcionalidades complejas donde el conocimiento sobre computación es muy importante.

11.2 Conclusiones

Este trabajo me ha ayudado a introducirme en el mundo laboral y experimentar el desarrollo de un proyecto de software en un caso real.

Primero de todo, es muy difícil gestionar un proyecto con poca experiencia y con la multitud de obstáculos que se presentan por el camino. El planteamiento inicial del trabajo junto a la ayuda de la empresa parecía razonable y fue muy motivador implementar nuevas funcionalidades en una aplicación donde usaría conocimientos avanzados sobre computación.

Sin embargo, ha surgido un gran imprevisto que ha resquebrajado la economía de la empresa obligando a dejar de lado la integración del proyecto en una plataforma que utilizan clientes. Al principio fue desmotivador pero se han encontrado soluciones efectivas que muestran el potencial del trabajo realizado.

El objetivo global de este proyecto era analizar de forma automática entrenamientos para poder evaluar el estado de forma de un deportista y conocer la tendencia de su rendimiento en un futuro. Después de finalizarlo puedo afirmar que se ha conseguido exitosamente además de construirse una buena base para poder escalar e integrar el proyecto. A pesar de dejar de lado algunas funcionalidades iniciales, lo considero una iteración que ha formado parte de la transformación del trabajo gravemente marcada por el efecto de los obstáculos. Ahora mismo se dispone de un aplicativo web que se ejecuta en local con estilo de diseño *dashboard* que permite evaluar y predecir el estado de forma de diferentes ciclistas automáticamente. Además, se ha conseguido mediante el desarrollo exitoso de diferentes modelos de aprendizaje supervisado.

Para acabar, he llegado a la conclusión final de que desarrollar un proyecto es un desafío y que es muy importante identificar detalles para hacerlos bien. Porque si nunca aprendes a hacer pequeñas cosas bien nunca podrás lograr grandes hazañas. Estoy muy agradecido de mi esfuerzo a pesar de las adversidades y de haber completado un reto enorme, ha sido una buena experiencia que me da pie a empezar nuevos proyectos.

12. Bibliografía y referencias

[1] "Consigue tu objetivo con Trainerer", *trainerer.com*, 2020 [En línea]. Disponible en: https://www.trainerer.com/consigue-tu-objetivo/ [Accedido: 10 May 2020]

[2] "Strava Features", *strava.com*, 2020 [En línea]. Disponible en: https://www.strava.com/features [Accedido: 12 May 2020]

[3] "Gitlab", *gitlab.com*, 2020 [En línea]. Disponible en: https://about.gitlab.com/ [Accedido: 13 May 2020]

[4] "Slack", *slack.com*, 2020 [En línea]. Disponible en: https://slack.com/intl/en-es/ [Accedido: 21 May 2020]

[5] "Scikit-Learn: Machine Learning in Python", *scikit-learn.org*, 2020 [En línea]. Disponible en: https://scikit-learn.org/stable/ [Accedido: 21 May 2020]

[6] "Gestión de proyectos Gantter", *chrome.google.com*, 2020 [En línea]. Disponible en: https://chrome.google.com/webstore/detail/gantter-project-managemen/himomacamcpodhkahelbnmaddladgjgo [Accedido: 21 May 2020]

[7] "What is MongoDB?", *mongodb.com*, 2020 [En línea]. Disponible en: https://www.mongodb.com/what-is-mongodb [Accedido: 21 May 2020]

[8] "MariaDB Server: The open source relational database", *mariadb.org*, 2020 [En línea]. Disponible en: https://mariadb.org [Accedido: 21 May 2020]

[9] "RabbitMQ", *rabbitmq.org*, 2020 [En línea]. Disponible en: https://www.rabbitmq.com/ [Accedido: 21 May 2020]

[10] "Visual Studio", *visualstudio.microsoft.com*, 2020 [En línea]. Disponible en: https://visualstudio.microsoft.com/ [Accedido: 21 May 2020]

[11] "Robo 3T", *robomongo.org*, 2020 [En línea]. Disponible en: https://robomongo.org/ [Accedido: 21 May 2020]

[12] "Matplotlib: Visualization with Python", *matplotlib.org*, 2020[En línea]. Disponible en: https://matplotlib.org/ [Accedido: 22 May 2020]

[13] "Project Jupyter", *jupyter.org*, 2020 [En línea]. Disponible en: https://jupyter.org/ [Accedido: 27 May 2020]

[14] "Salary Data and Career Research Center (Spain)", *payscale.com*, 2020 [En línea]. Disponible en: https://www.payscale.com/research/ES/Country=Spain/Salary [Accedido: 27 May 2020]

[15] "Analizamos el FTP (umbral funcional de potencia)", *adnciclista.com*, 2020 [En línea]. Disponible en: https://www.adnciclista.com/ftp-umbral-funcional-potencia/ [Accedido: 30 May 2020]

[16] “¿Qué es “cadencia” en el ciclismo?”, *g-se.com*, 2020 [En línea]. Disponible en: https://g-se.com/que-es-cadencia-en-el-ciclismo-bp-o57cfb26de3d96 [Accedido: 30 May 2020]

[17] “Habla de potencia y vatios con propiedad”, *ciclismoafondo.es*, 2016 [En línea]. Disponible en: https://www.ciclismoafondo.es/preparacion-fisica/entrenamiento-ciclismo/articulo/explicacion-terminos-potencia-vatios-potenciometros#pagina-2 [Accedido: 30 May 2020]

[18] "Gradients And Cycling: An Introduction", *theclimbingcyclist.com*, 2013 [En línea]. Disponible en: http://theclimbingcyclist.com/gradients-and-cycling-an-introduction/ [Accedido: 29 May 2020]

[19] "What is Docker? How does it work?", *devopscube.com*, 2020 [En línea]. Disponible en: https://devopscube.com/what-is-docker/ [Accedido: 3 Sep 2020]

[20] "Introduction to Flask", *pymbook.readthedocs.io*, 2020 [En línea]. Disponible en: https://pymbook.readthedocs.io/en/latest/flask.html [Accedido: 3 Sep 2020]

[21] “What is the difference between ODM and ORM?”, *medium.com*, 2020 [En línea]. Disponible en: https://medium.com/@julianam.tyler/what-is-the-difference-between-odm-and-orm-267bbb7778b0 [Accedido: 26 May 2020]

[22] “Humble DB: MongoDB micro-ODM”, *humbledb.readthedocs.io*, 2020 [En línea]. Disponible en: https://humbledb.readthedocs.io/en/latest/ [Accedido: 26 May 2020]

[23] “The Python SQL Toolkit and Object Relational Mapper”, *sqlalchemy.org*, 2020 [En línea]. Disponible en: https://www.sqlalchemy.org/ [Accedido: 26 May 2020]

[24] "Linear Regression in Python", *realpython.com*, 2020 [En línea]. Disponible en: https://realpython.com/linear-regression-in-python/ [Accedido: 3 Sep 2020]

[25] “K-Nearest Neighbors (kNN)”, *towardsdatascience.com*, 2020 [En línea]. Disponible en: https://towardsdatascience.com/k-nearest-neighbors-knn-explained-cbc31849a7e3 [Accedido: 14 Sep 2020]

[26] Gonzalo Berástegui Arbeloa, Mikel Galar Idoate, “Implementación del algoritmo de los k vecinos más cercanos (k-NN) y estimación del mejor valor local de k para su cálculo”, Trabajo de Fin de Grado, Universidad Pública de Navarra. Marzo 2018.

[27] “Random Forest Regression”, *towardsdatascience.com*, 2019 [En línea]. Disponible en: https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f [Accedido: 14 Sep 2020]

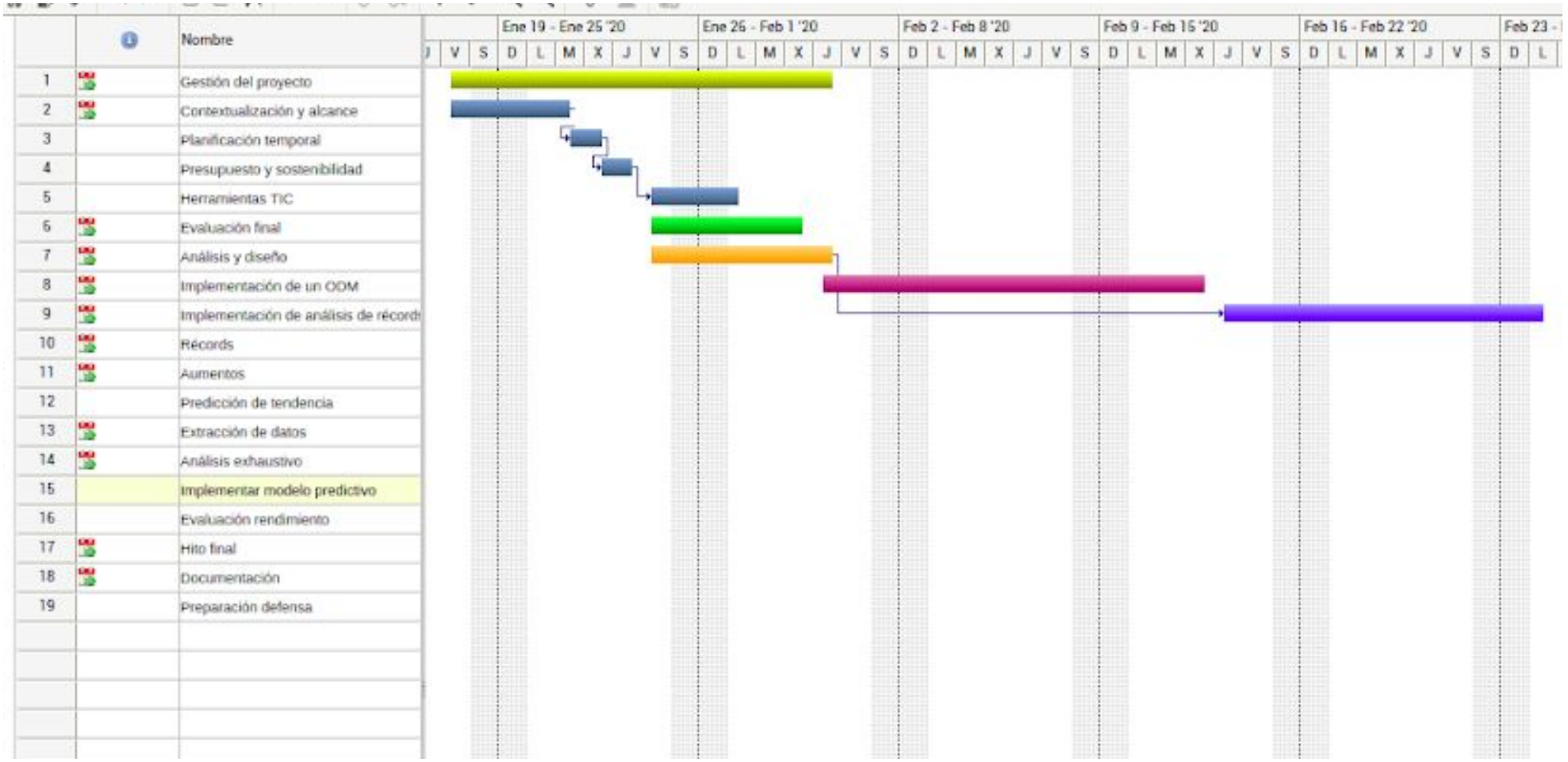
[28] “How to Convert a Time Series to a Supervised Learning Problem in Python”, *machinelearningmastery.com*, 2017 [En línea]. Disponible en: https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/ [Accedido: 15 Sep 2020]

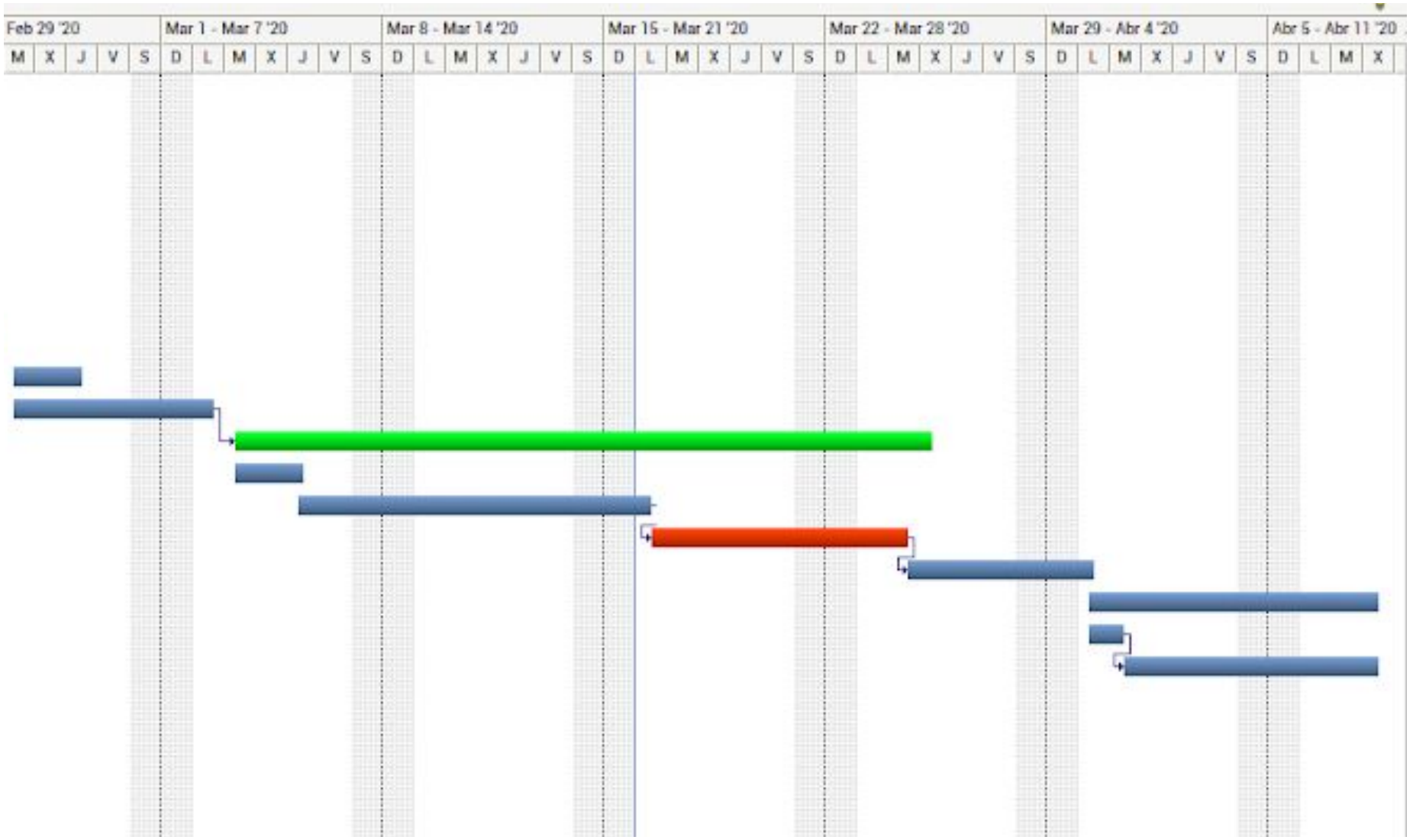
[29] “pandas.DataFrame.shift”, *pandas.pydata.org*, 2020 [En línea]. Disponible en: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.shift.html [Accedido: 13 Sep 2020]

[30] “How to Develop Multi-Output Regression Models with Python”, *machinelearningmastery.com*, 2020 [En línea]. Disponible en: https://machinelearningmastery.com/multi-output-regression-models-with-python/ [Accedido: 14 Sep 2020]

[31] "What does RMSE really mean?", *towardsdatascience.com*, 2019 [En línea]. Disponible en: https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e [Accedido: 18 Sep 2020]

Anexo 1. Diagrama de Gantt.




Feb 29 '20
Mar 1 - Mar 7 '20
Mar 8 - Mar 14 '20
Mar 15 - Mar 21 '20
Mar 22 - Mar 28 '20
Mar 29 - Abr 4 '20
Abr 5 - Abr 11 '20
M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X

Anexo 2. Análisis estadístico de ciclismo en ruta

PreprocessingCyclingStreet

October 22, 2020

1 Análisis estadístico del deporte CYCLING_STREET

```
In [3]: # Actualizar packages
        !pip3 install pandas --user --upgrade --quiet
        !pip3 install numpy --user --upgrade --quiet
        !pip3 install scipy --user --upgrade --quiet
        !pip3 install statsmodels --user --upgrade --quiet
        %load_ext autoreload
```

```
In [1]: #%matplotlib notebook
        %reload_ext autoreload
        import numpy as np
        import matplotlib.pyplot as plt
        #import seaborn as sn
        import pandas as pd
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
        pd.set_option('precision', 3)
```

```
In [33]: # extra imports
         from pandas import read_csv
         #from sklearn.neighbors import KNeighborsClassifier
         from statsmodels.genmod.generalized_linear_model import GLM
         from pandas.plotting import scatter_matrix
         from scipy.stats import boxcox, shapiro, normaltest
         import math
         from statsmodels.graphics.gofplots import qqplot
```

```
In [3]: Zones = read_csv("dataCYCLING_STREET.csv", header=0, delimiter=',')
        Zones.shape
```

```
Out[3]: (4303, 9)
```

```
In [4]: Zones.columns
```

```
Out[4]: Index(['Date', 'DurationSecondsHR', 'VelocityHR', 'PotencyHR', 'GradientHR',
               'DurationSecondsPOWER', 'VelocityPOWER', 'PotencyPOWER',
               'GradientPOWER'],
              dtype='object')
```

```
In [5]: Zones[:4]
```

```
Out[5]:    Date  DurationSecondsHR  VelocityHR  PotencyHR  GradientHR  \
        0    12             4869.0       7.741        NaN       0.694
        1   106              797.0       5.097        NaN       3.669
        2   103              680.0       1.827        NaN      10.685
        3   100              355.0       6.101        NaN       3.008

           DurationSecondsPOWER  VelocityPOWER  PotencyPOWER  GradientPOWER
        0                   NaN            NaN           NaN            NaN
        1                   NaN            NaN           NaN            NaN
        2                   NaN            NaN           NaN            NaN
        3                   NaN            NaN           NaN            NaN
```

1.1 Inspección básica de los datos

```
In [6]: Zones.describe()
```

```
Out[6]:             Date  DurationSecondsHR  VelocityHR  PotencyHR  GradientHR  \
        count  4303.000           3981.000    3981.000    818.000    3943.000
        mean   1054.611            605.907       5.651    251.440       5.579
        std     496.001            719.065       2.737     48.344      40.022
        min       1.000             60.000       0.000      0.000       0.000
        25%     657.000            199.000       3.674    220.948       1.995
        50%    1108.000            408.000       5.397    250.095       4.398
        75%    1483.000            769.000       7.205    288.309       6.447
        max    1760.000          14347.000      58.964    377.928    2423.529

               DurationSecondsPOWER  VelocityPOWER  PotencyPOWER  GradientPOWER
        count               1123.000       1123.000      1123.000       1080.000
        mean                 683.986          6.793       259.688          3.418
        std                  636.842          2.268        49.272          2.411
        min                   60.000          0.000        88.471          0.000
        25%                  221.000          5.178       220.936          1.399
        50%                  501.000          6.402       262.933          3.234
        75%                  903.500          7.991       292.154          4.976
        max                 5406.000         15.924       515.653         21.434
```

Hay valores desaparecidos por algun cero en el mínimo que es sospechoso.

```
In [7]: (Zones.VelocityHR==0).value_counts()
        (Zones.PotencyHR==0).value_counts()
        (Zones.VelocityPOWER==0).value_counts()
        #(Zones.DurationSecondsHR=='NaN').value_counts()
        #(Zones.VelocityHR=='NaN').value_counts()
```

```
Out[7]: False    4298
        True        5
        Name: VelocityHR, dtype: int64
```

```
Out[7]: False    4299
        True        4
        Name: PotencyHR, dtype: int64

Out[7]: False    4302
        True        1
        Name: VelocityPOWER, dtype: int64
```

A continuación se examinan cuántos valores son null. No es necesariamente un error.

```
In [8]: (Zones.Date.isnull()).value_counts()
        (Zones.DurationSecondsHR.isnull()).value_counts()
        (Zones.VelocityHR.isnull()).value_counts()

Out[8]: False    4303
        Name: Date, dtype: int64

Out[8]: False    3981
        True      322
        Name: DurationSecondsHR, dtype: int64

Out[8]: False    3981
        True      322
        Name: VelocityHR, dtype: int64
```

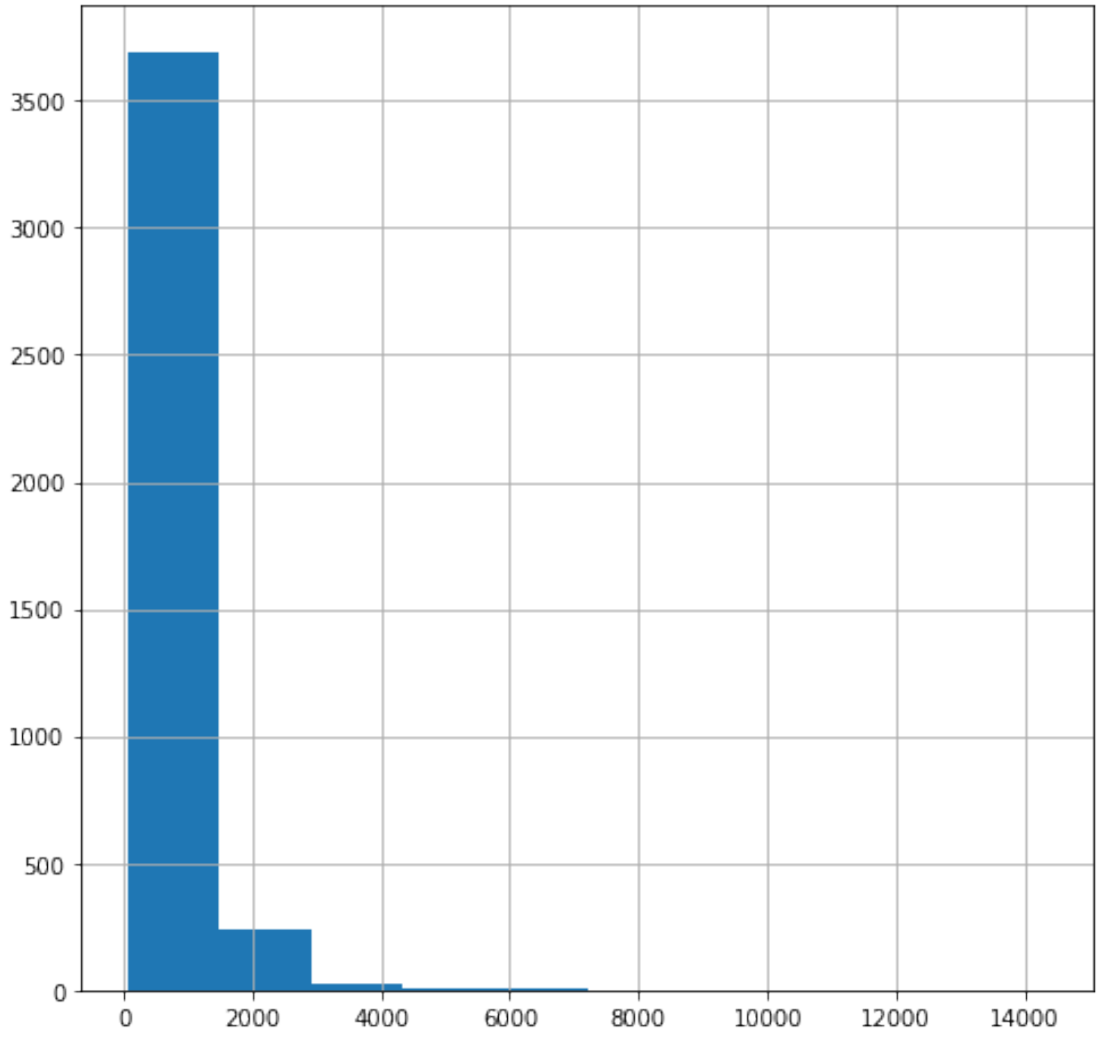
Valores que faltan relacionados con la potencia.

```
In [9]: (Zones.PotencyHR.isnull()).value_counts()
        (Zones.DurationSecondsPOWER.isnull()).value_counts()
        (Zones.VelocityPOWER.isnull()).value_counts()
        (Zones.PotencyPOWER.isnull()).value_counts()
        (Zones.GradientPOWER.isnull()).value_counts()

Out[9]: True     3485
        False     818
        Name: PotencyHR, dtype: int64

Out[9]: True     3180
        False    1123
        Name: DurationSecondsPOWER, dtype: int64

Out[9]: True     3180
        False    1123
        Name: VelocityPOWER, dtype: int64

Out[9]: True     3180
        False    1123
        Name: PotencyPOWER, dtype: int64
```

```
Out[9]: True     3223
        False    1080
        Name: GradientPOWER, dtype: int64
```

Como se comenta en el apartado correspondiente de la parte principal de la memoria, esto es debido a que las actividades registradas vienen de bicicletas sin potenciómetro para registrar información para calcular estos parámetros.

```
In [10]: Zones.DurationSecondsHR.hist(figsize=(8,8));
```



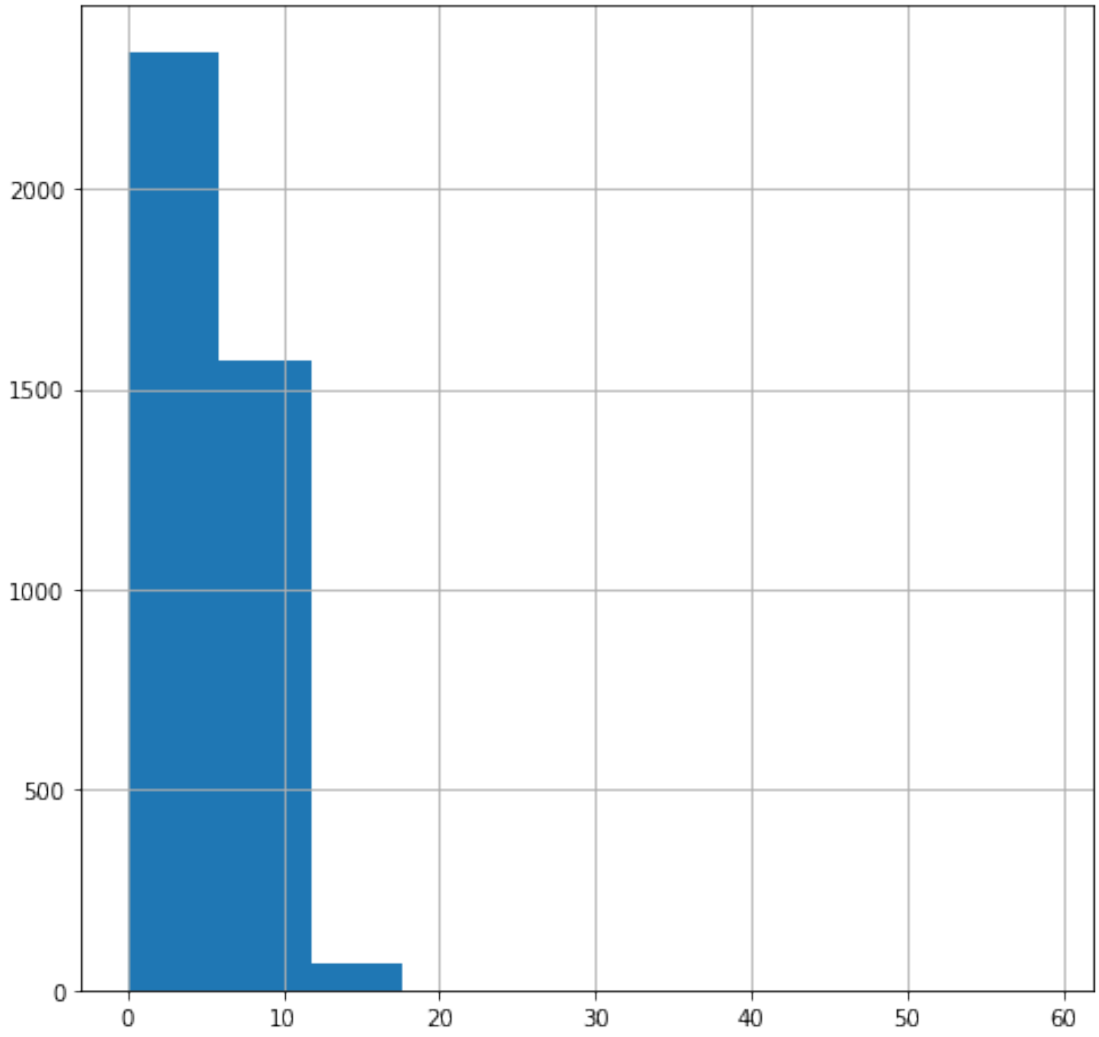
No se aprecian outlayers pero estan en el gráfico, parace que estan a partir de 8000.

```
In [11]: (Zones.DurationSecondsHR>8000).value_counts()
```

```
Out[11]: False    4299
         True        4
         Name: DurationSecondsHR, dtype: int64
```

4 outlayers en DurationSecondsHR.
Lo mismo para los otros parámetros:

```
In [12]: Zones.VelocityHR.hist(figsize=(8,8));
```



Test de normalidad Shapiro-Wilk para esta variable:

```
In [27]: shapiro_test = shapiro(Zones.VelocityHR.dropna())
         shapiro_test
```

```
Out[27]: ShapiroResult(statistic=0.90326327085495, pvalue=1.1210387714598537e-44)
```

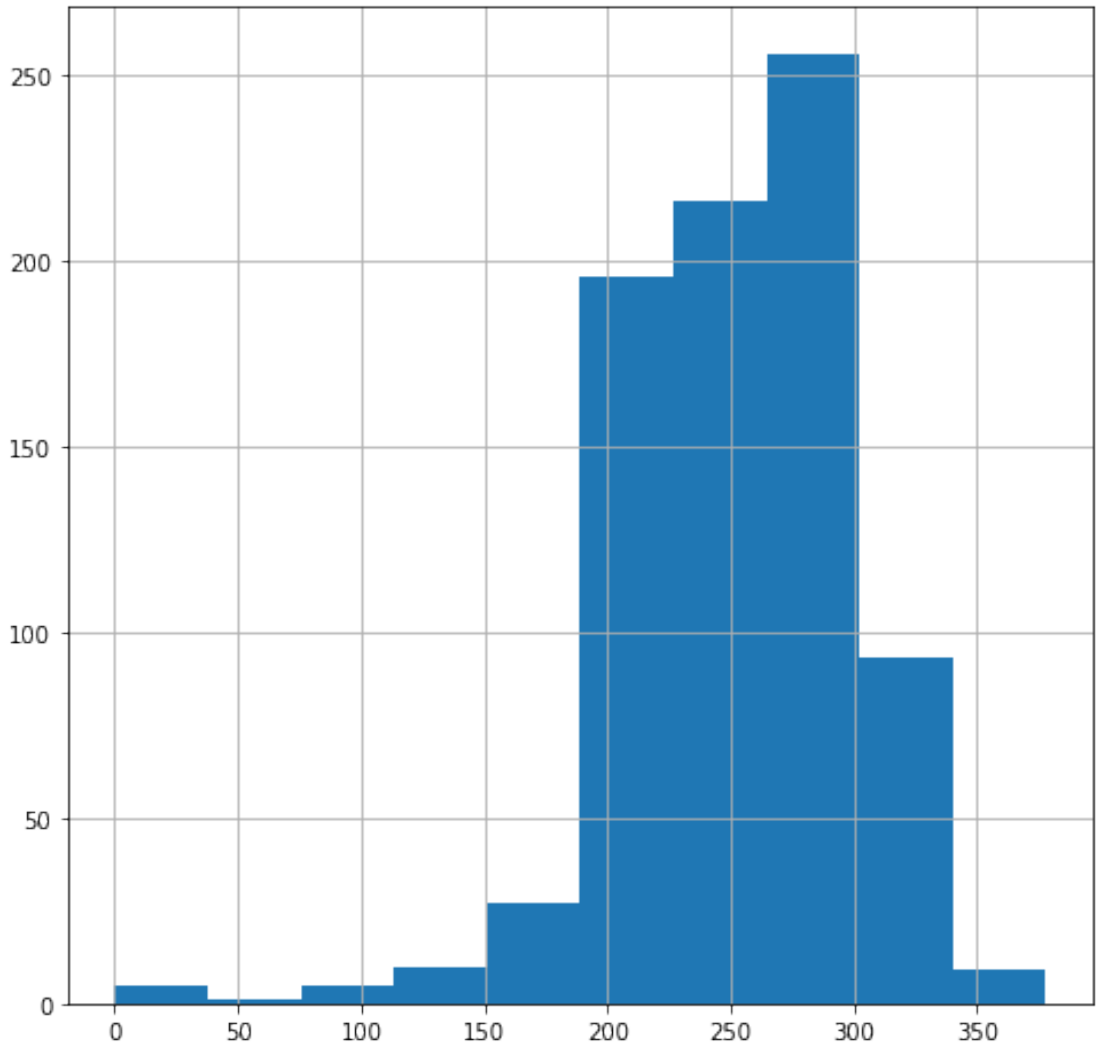
Como el p-value está por debajo del intervalo de confianza 0.05 los datos no siguen una distribución Gaussiana. Por otro lado, los datos siguen una distribución unimodal y es una propiedad muy interesante para poder hacer un filtrado de ruido en los datos a través de por ejemplo la desviación estándar.

En resumen, los datos de este parámetro tienen muy buena forma. Se diferencia claramente la parte donde están los outlayers.

```
In [13]: (Zones.VelocityHR>20).value_counts()

Out[13]: False    4301
         True        2
         Name: VelocityHR, dtype: int64
```

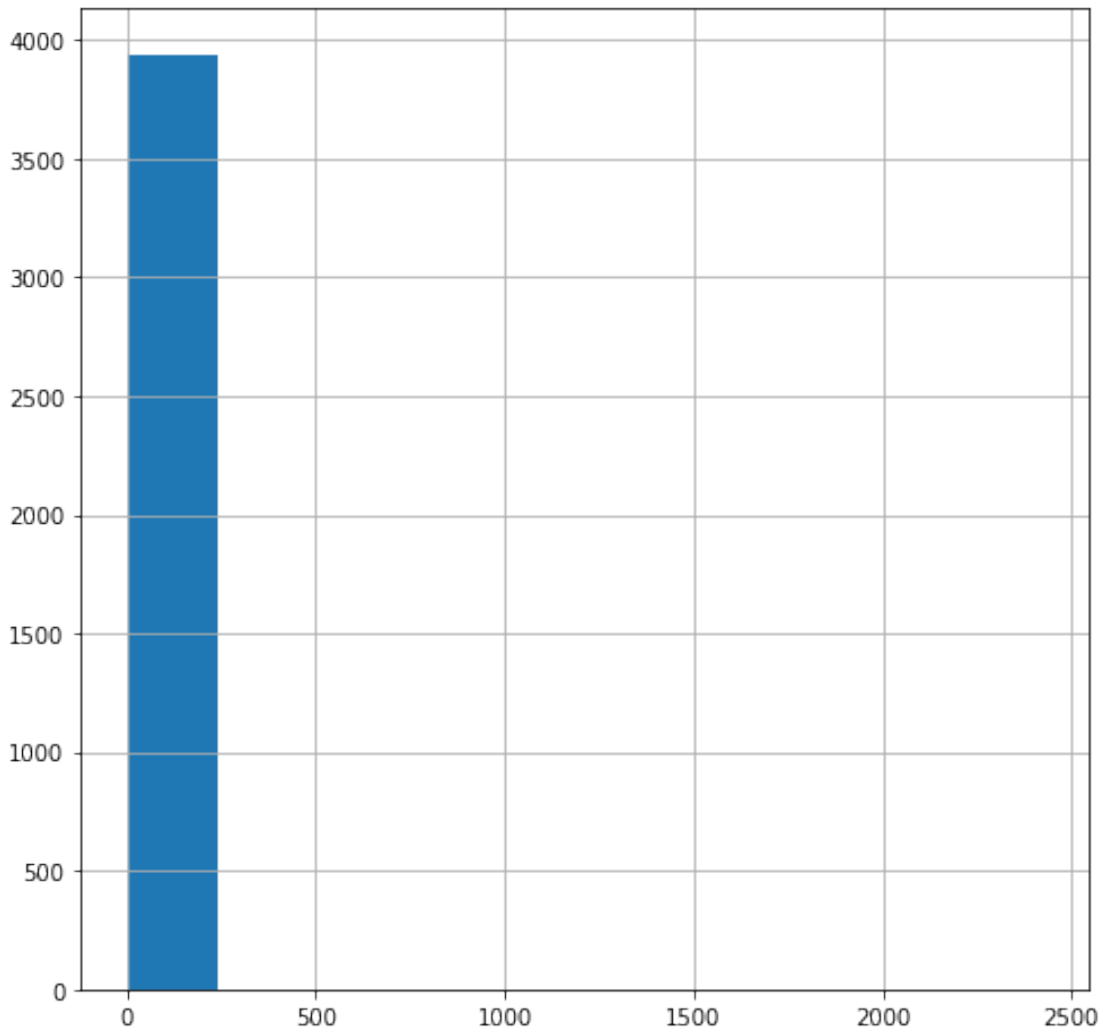
Hay 2 outlayers en VelocityHR.

```
In [14]: Zones.PotencyHR.hist(figsize=(8,8));
```

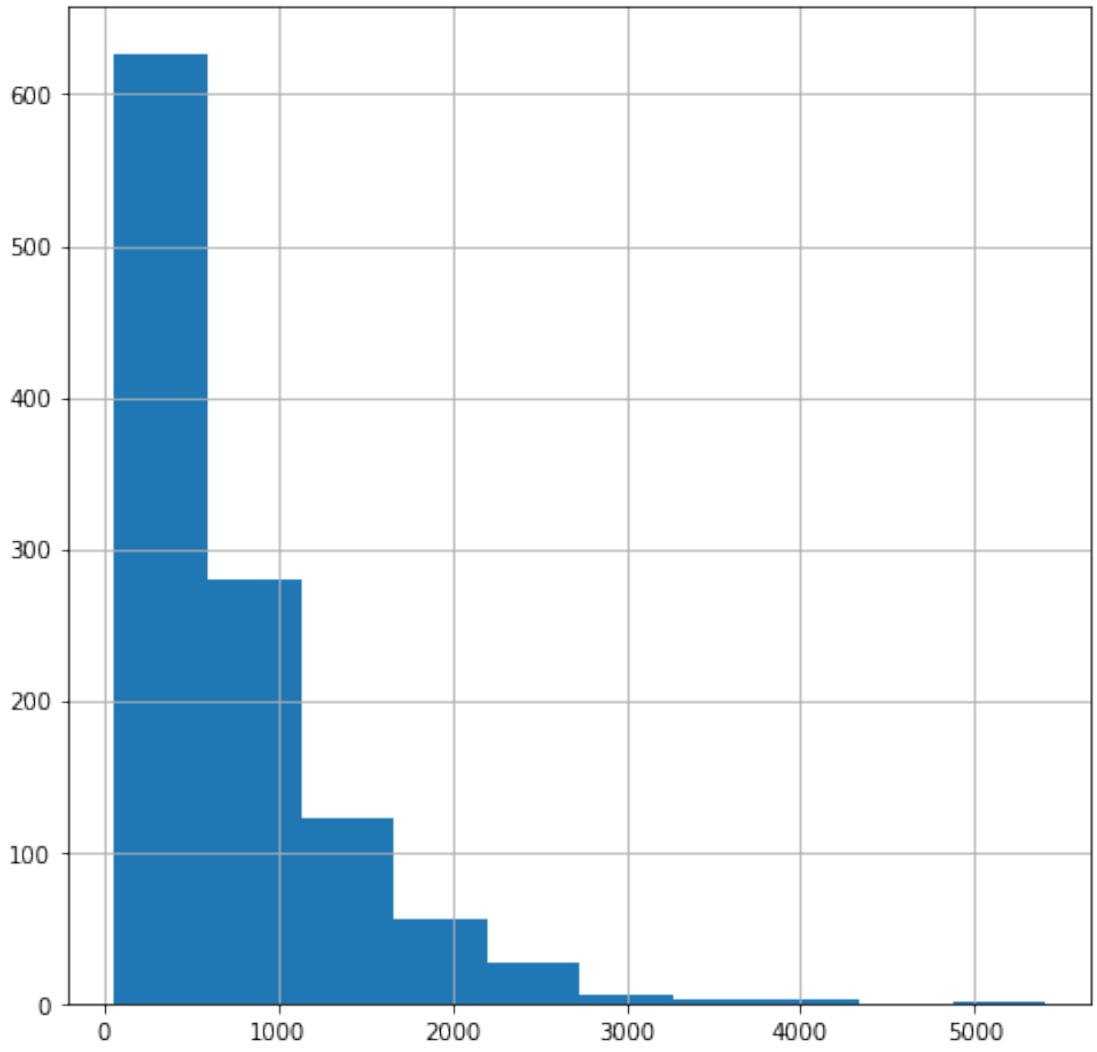


Los valores cercanos a 0 no son necesariamente outlayers o valores desaparecidos porque se puede dar el caso de registrar parámetros cuando no se está produciendo ninguna potencia durante el pedaleo como en un descenso.

```
In [15]: Zones.GradientHR.hist(figsize=(8,8));
```



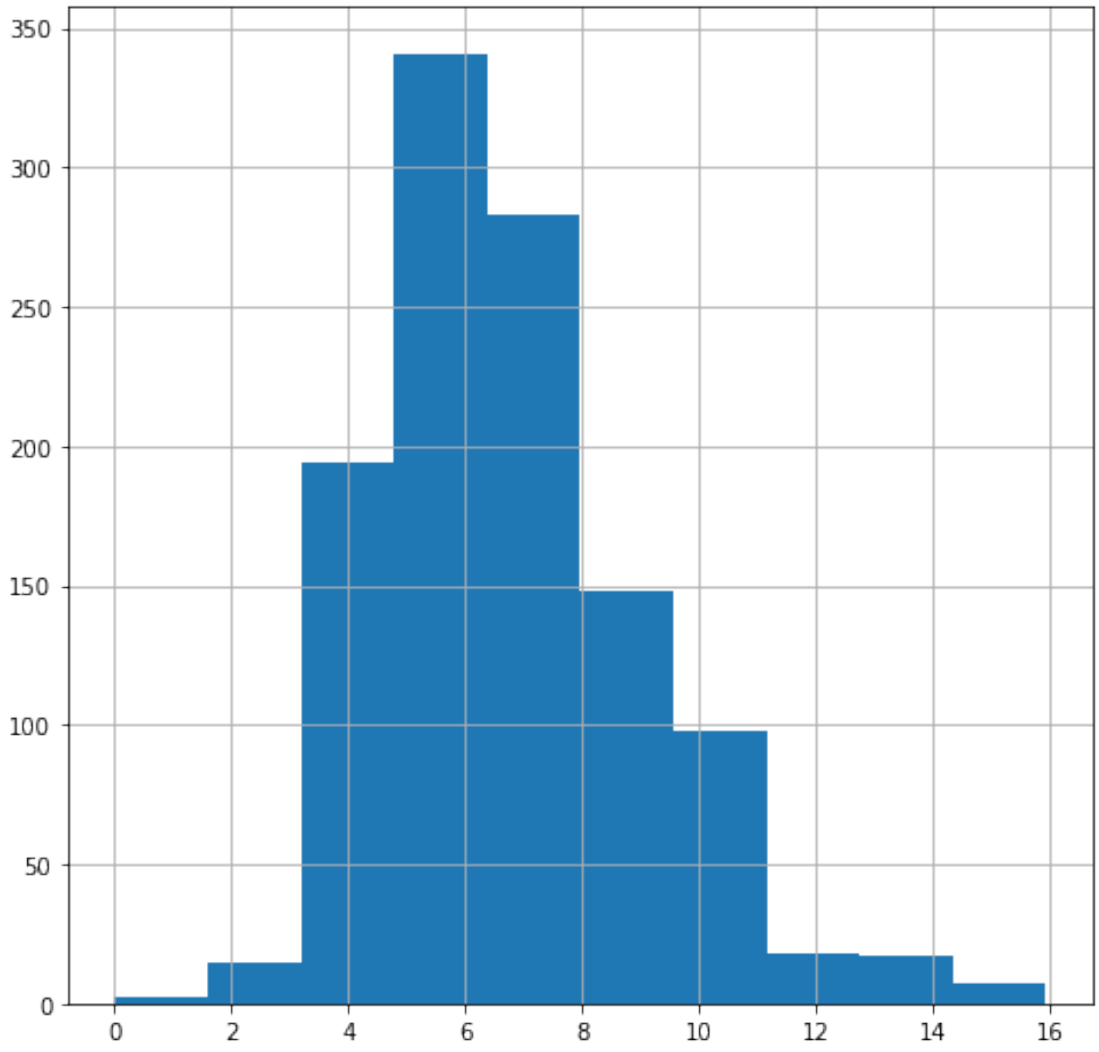

```
In [16]: (Zones.GradientHR>20).value_counts()

Out[16]: False    4291
         True       12
         Name: GradientHR, dtype: int64

In [17]: Zones.DurationSecondsPOWER.hist(figsize=(8,8));
```




```
In [18]: (Zones.DurationSecondsPOWER>4500).value_counts()

Out[18]: False    4302
         True        1
         Name: DurationSecondsPOWER, dtype: int64
```

1 outlayer en DurationSecondsPOWER.

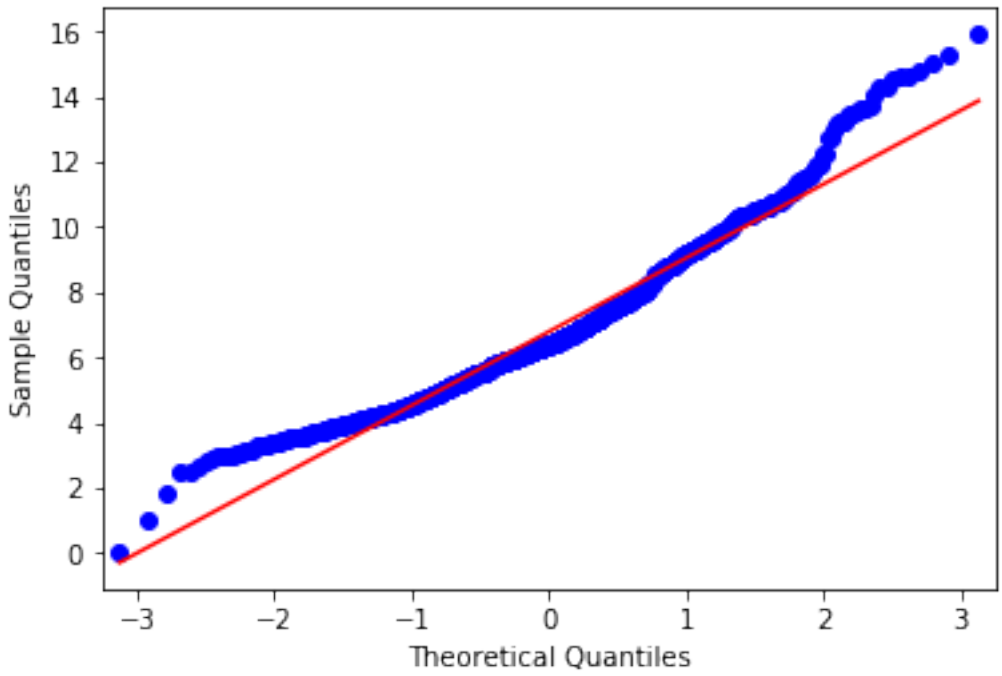
```
In [20]: Zones.VelocityPOWER.hist(figsize=(8,8));
```

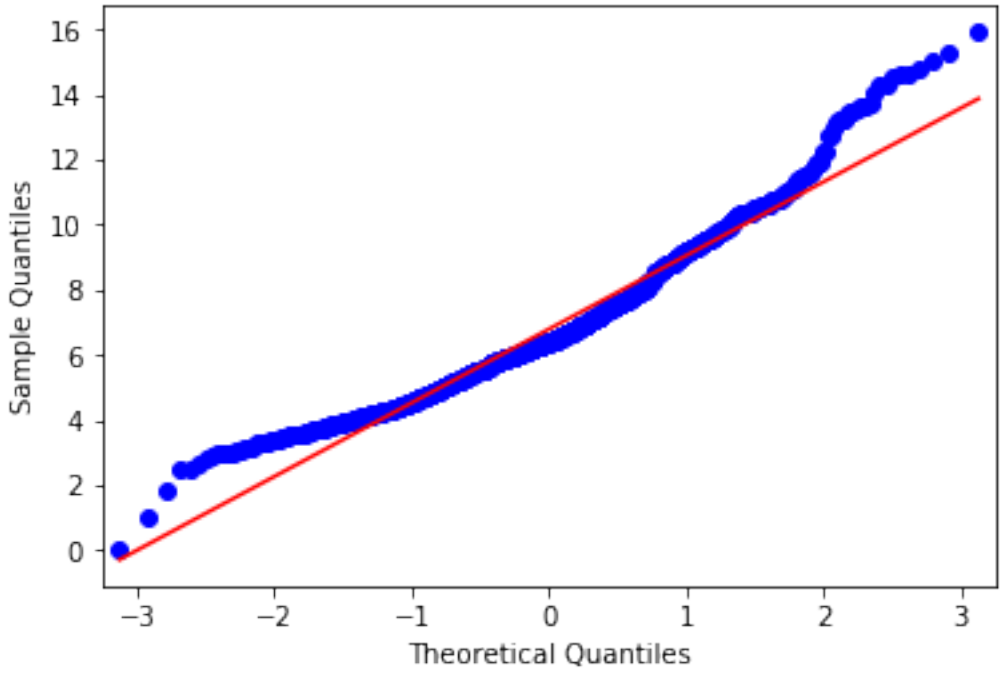


A simple vista, los datos de VelocityPOWER parece que son Gaussianos.

```
In [40]: data = Zones.VelocityPOWER.dropna()
         qqplot(data, line='s')
         plt.show()
```

Out[40]:





En estos gráficos, contra más se ajusten los datos azules a la línea diagonal roja significa que es más probable que sigan un patrón de una distribución Gaussiana. En este caso la parte central se ajusta, pero los extremos no cosa que no proporciona una conclusión clara.

```
In [44]: shapiro_test = shapiro(data)
         shapiro_test

Out[44]: ShapiroResult(statistic=0.9593892097473145, pvalue=3.9740205081660606e-17)
```

Procedemos a realizar un test de Shapiro-Wilk para tener una conclusión final. El p-value resultante ha salido mucho más pequeño que el valor aceptable de 0.05 por lo tanto la distribución de VelocityPOWER no es Gaussiana. Por otro lado y como en el otro caso, la distribución si es unimodal.

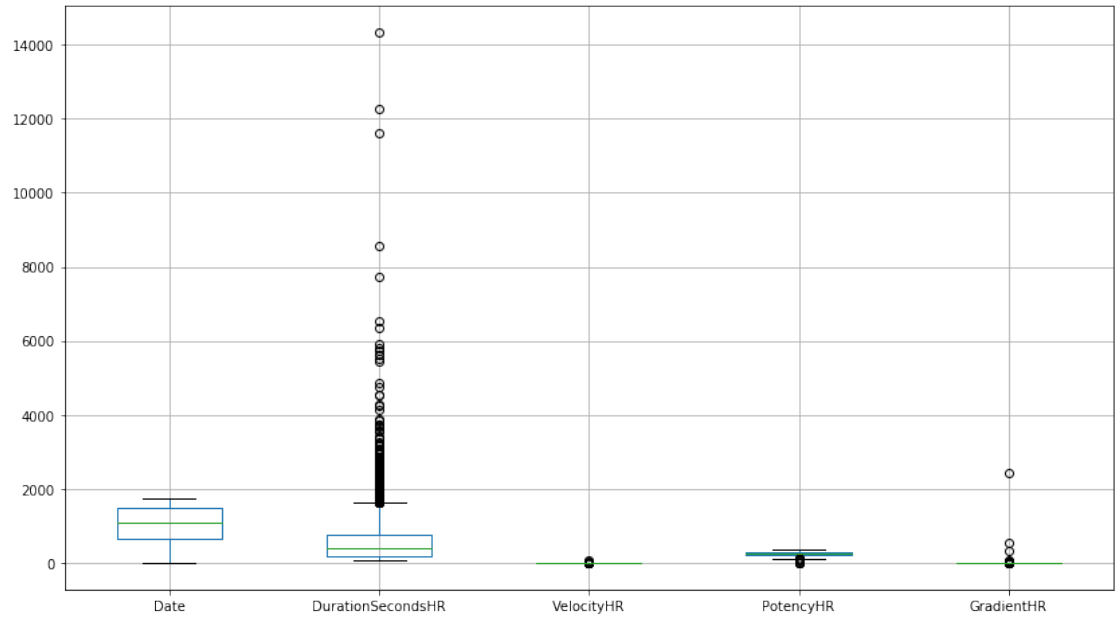
Aparentemente no hay ningún outlayer en VelocityPOWER observando su histograma.

1.2 Resumen gráfico

```
In [60]: #transforma a una distrubución gaussiana
         #pd.options.mode.use_inf_as_na = True
         #Zones.DurationSecondsHR[Zones.DurationSecondsHR<20000].apply(np.log10).hist(bins=20,
```

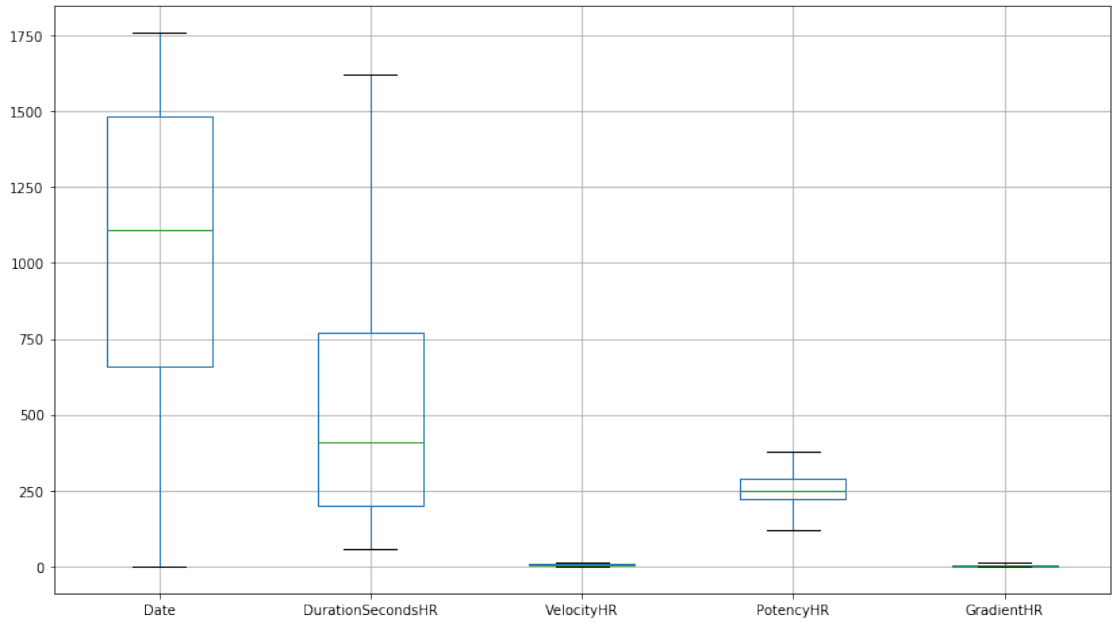
Vista general parámetros HR:

```
In [71]: Zones.loc[:,"Date":"GradientHR"].boxplot(figsize=(14,8));
```



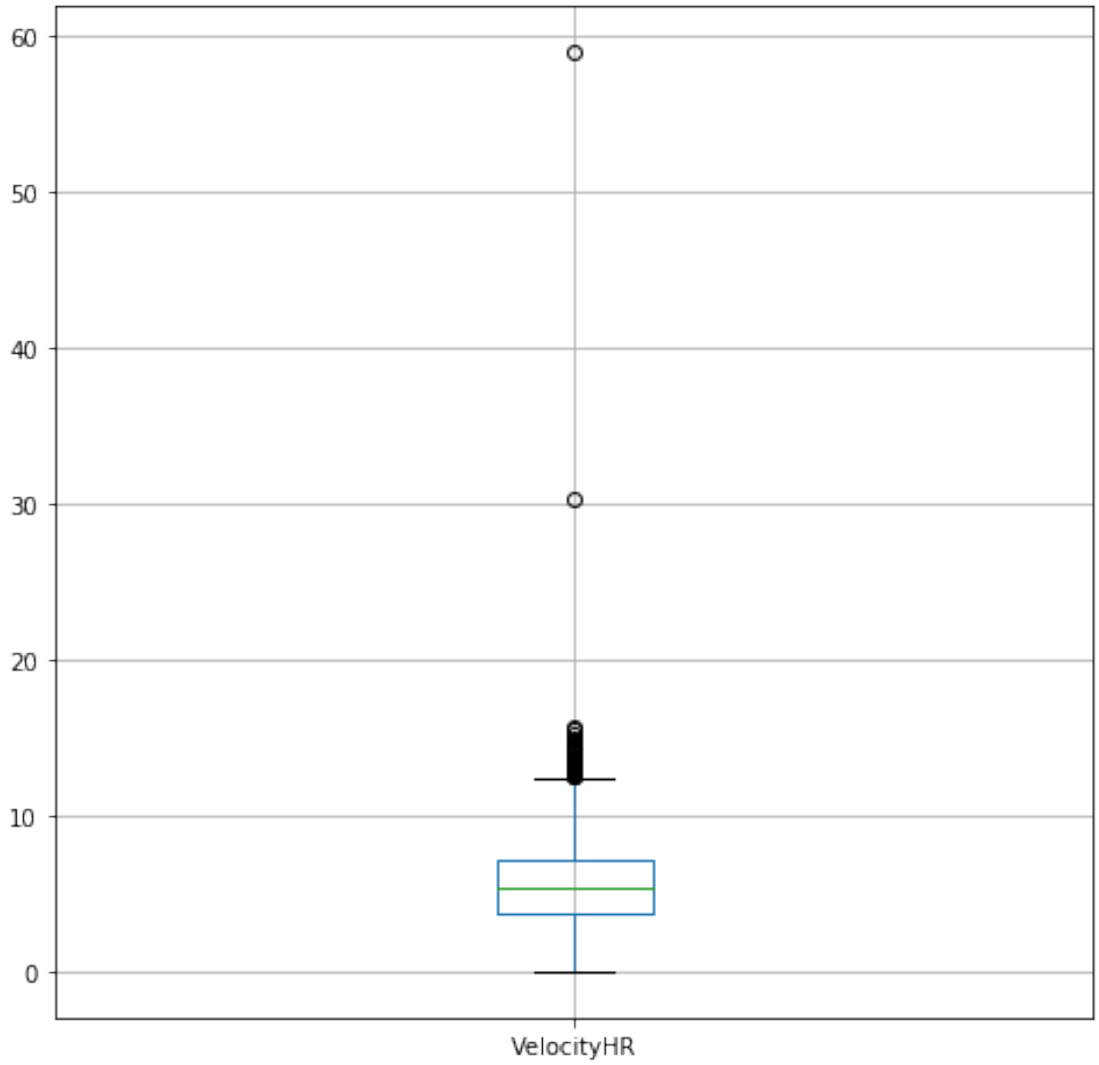
De forma más limpia:

```
In [72]: Zones.loc[:,"Date":"GradientHR"].boxplot(figsize=(14,8), showfliers=False);
```



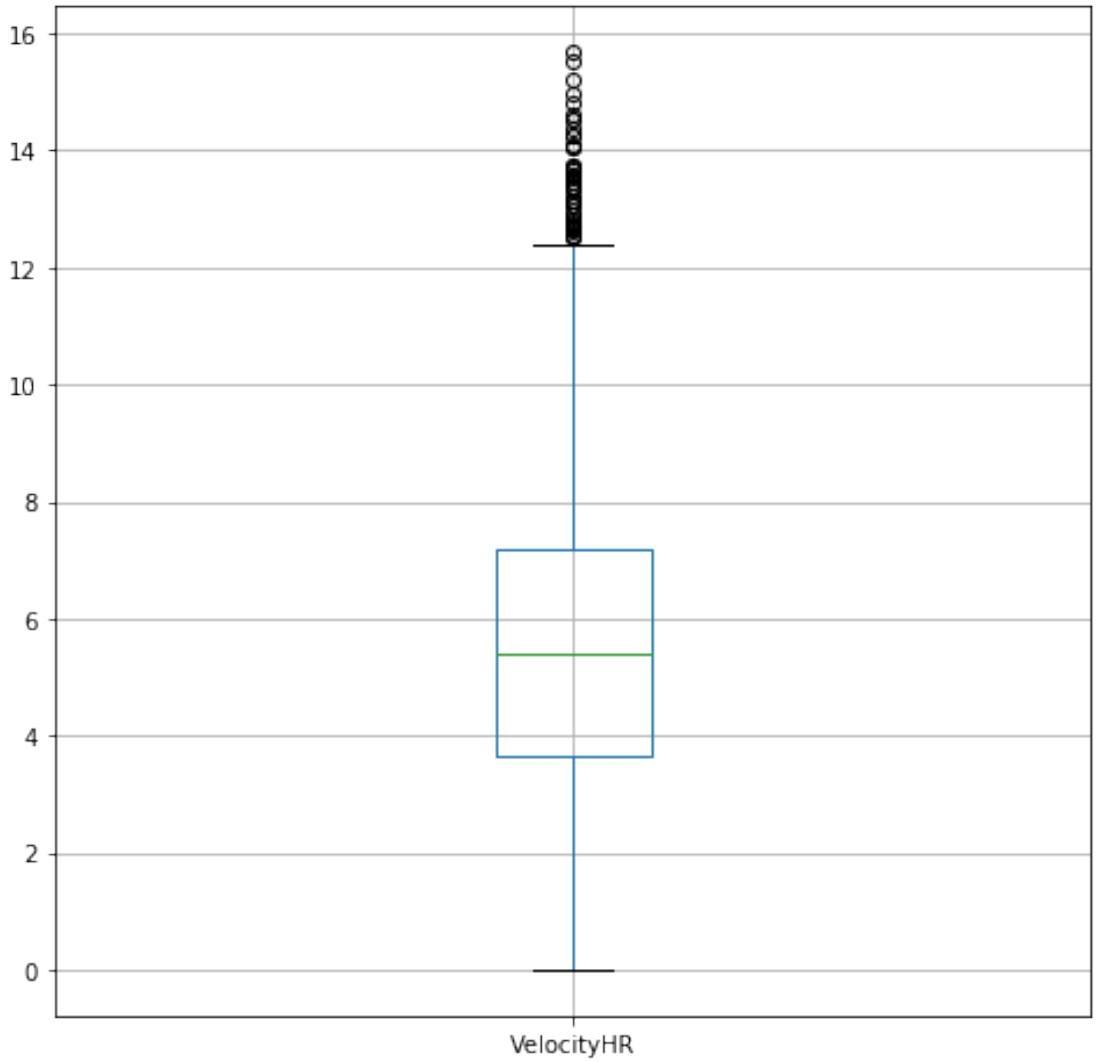
A continuación más en detalle velocityHR y gradientHR:

```
In [65]: Zones.boxplot(column='VelocityHR',figsize=(8,8));
```

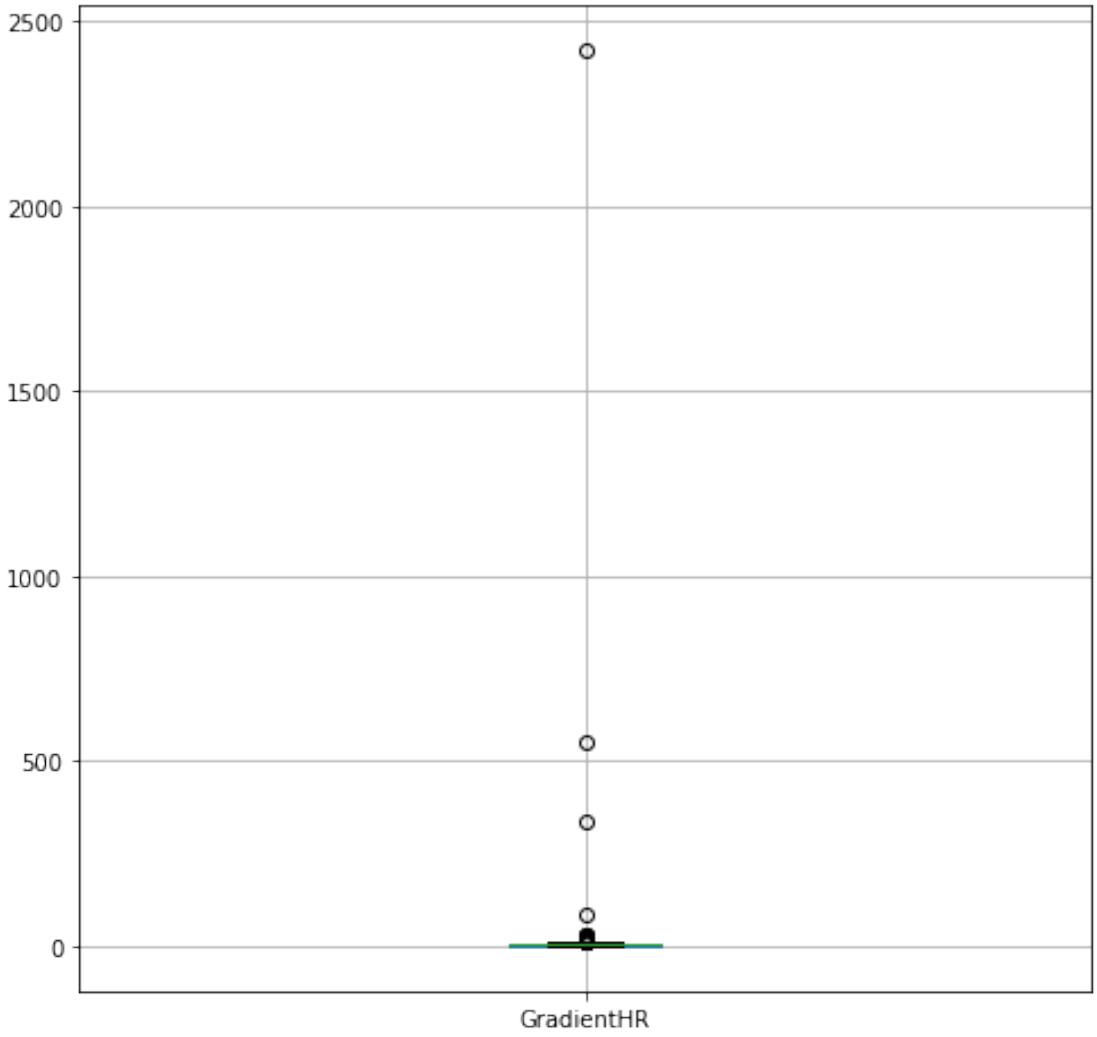


Limpiando outlayers:

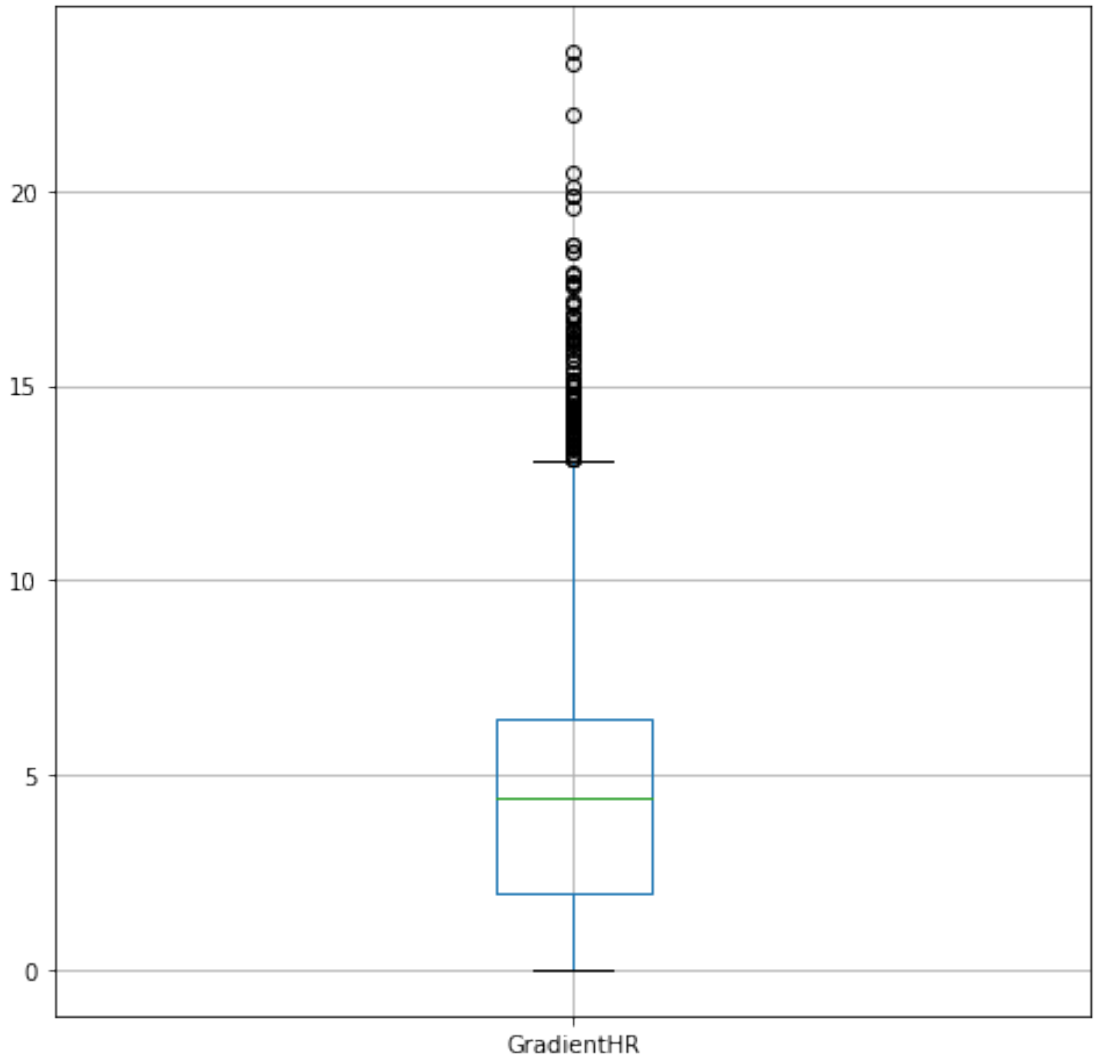
```
In [66]: Zones[Zones.VelocityHR<20].boxplot(column='VelocityHR',figsize=(8,8));
```



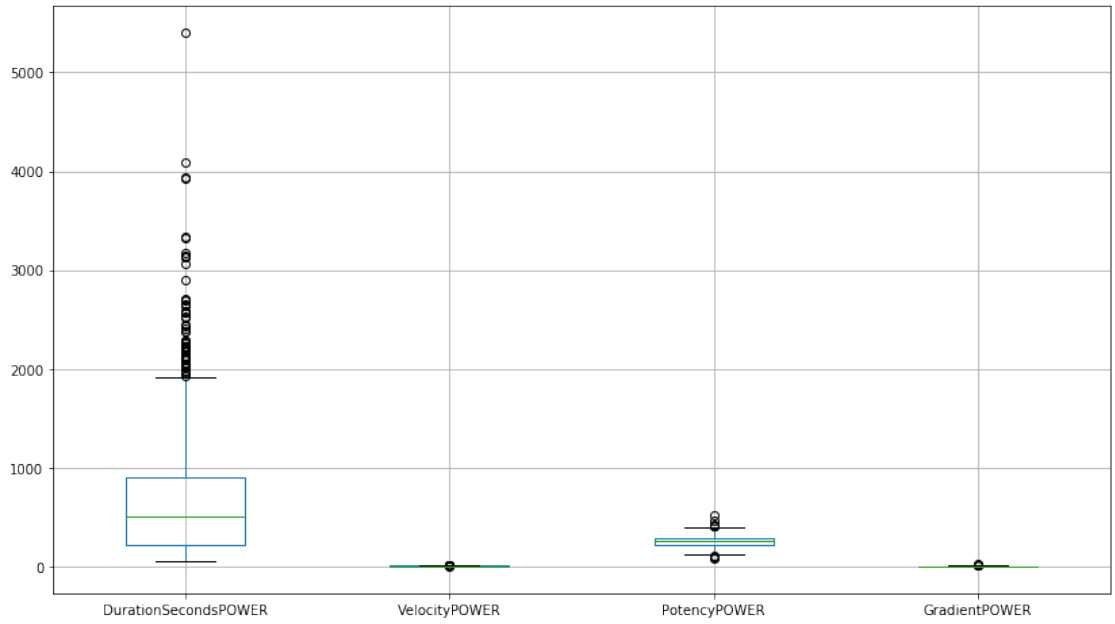
In [74]: Zones.boxplot(column='GradientHR',figsize=(8,8));



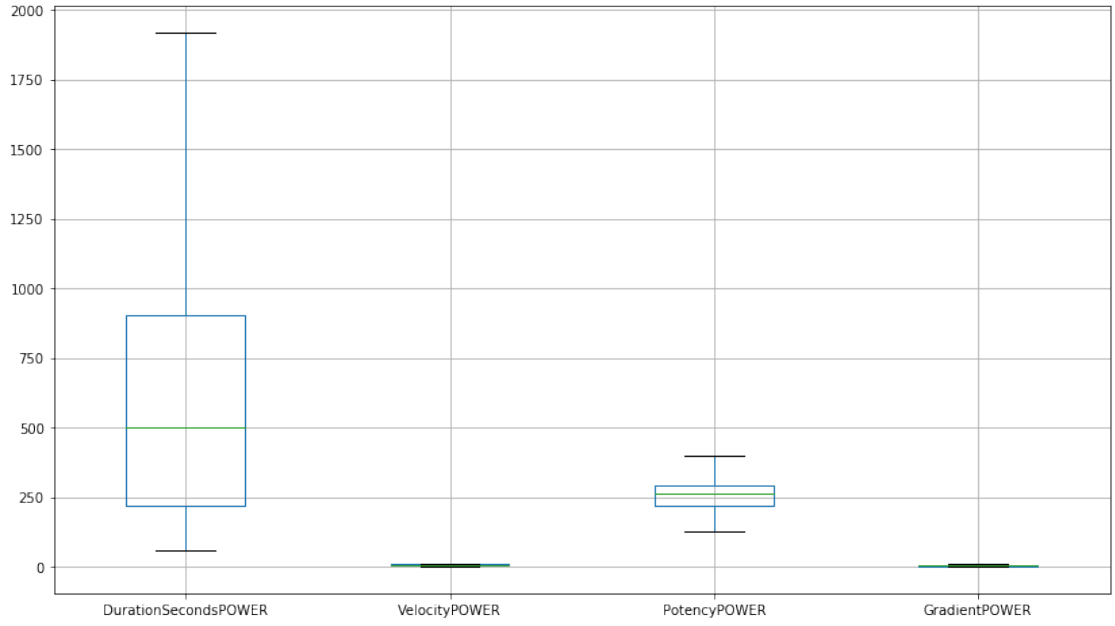

```
In [77]: Zones[Zones.GradientHR<25].boxplot(column='GradientHR',figsize=(8,8));
```




```
In [78]: Zones.loc[:,"DurationSecondsPOWER":"GradientPOWER"].boxplot(figsize=(14,8));
```

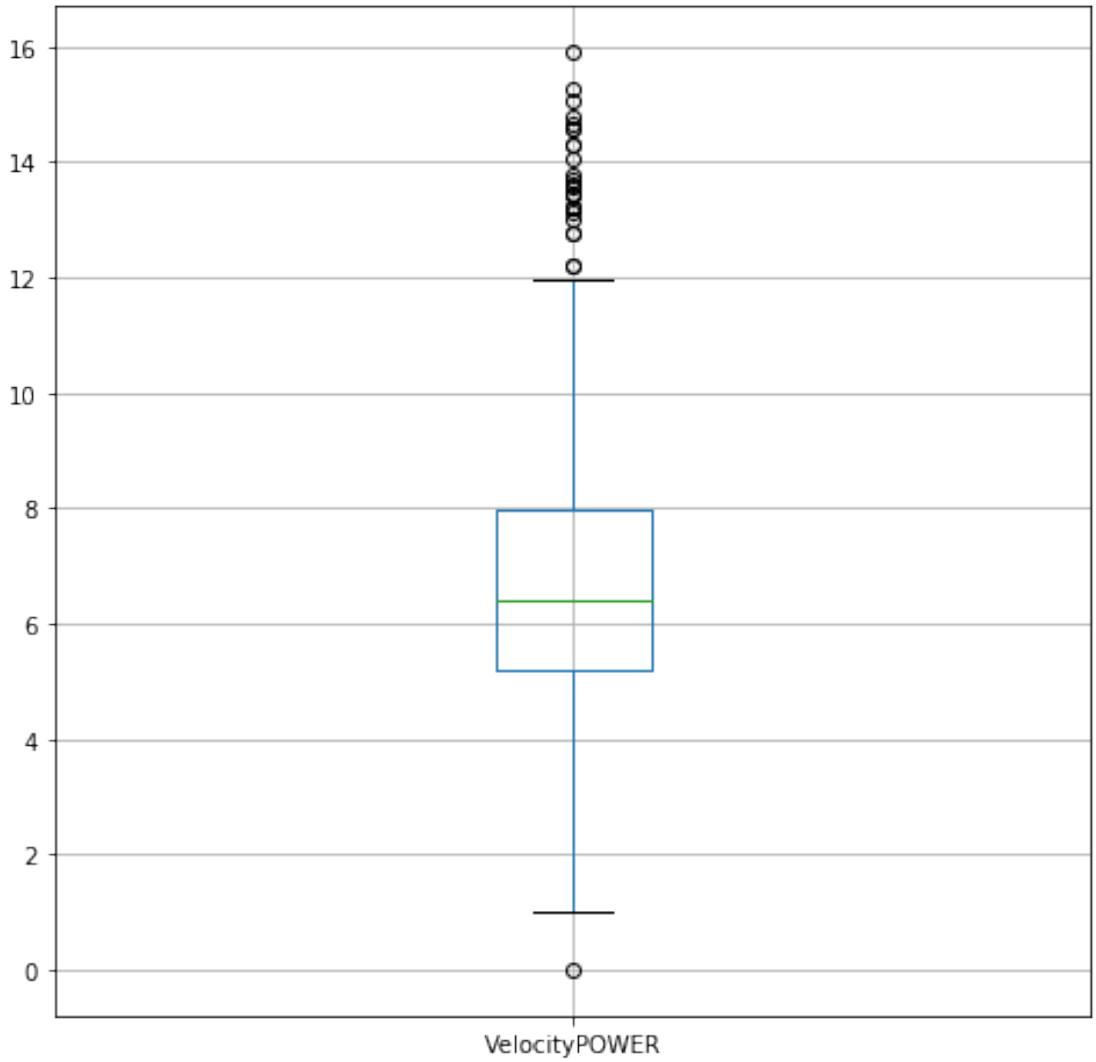



```
In [79]: Zones.loc[:,"DurationSecondsPOWER":"GradientPOWER"].boxplot(figsize=(14,8), showflier
```

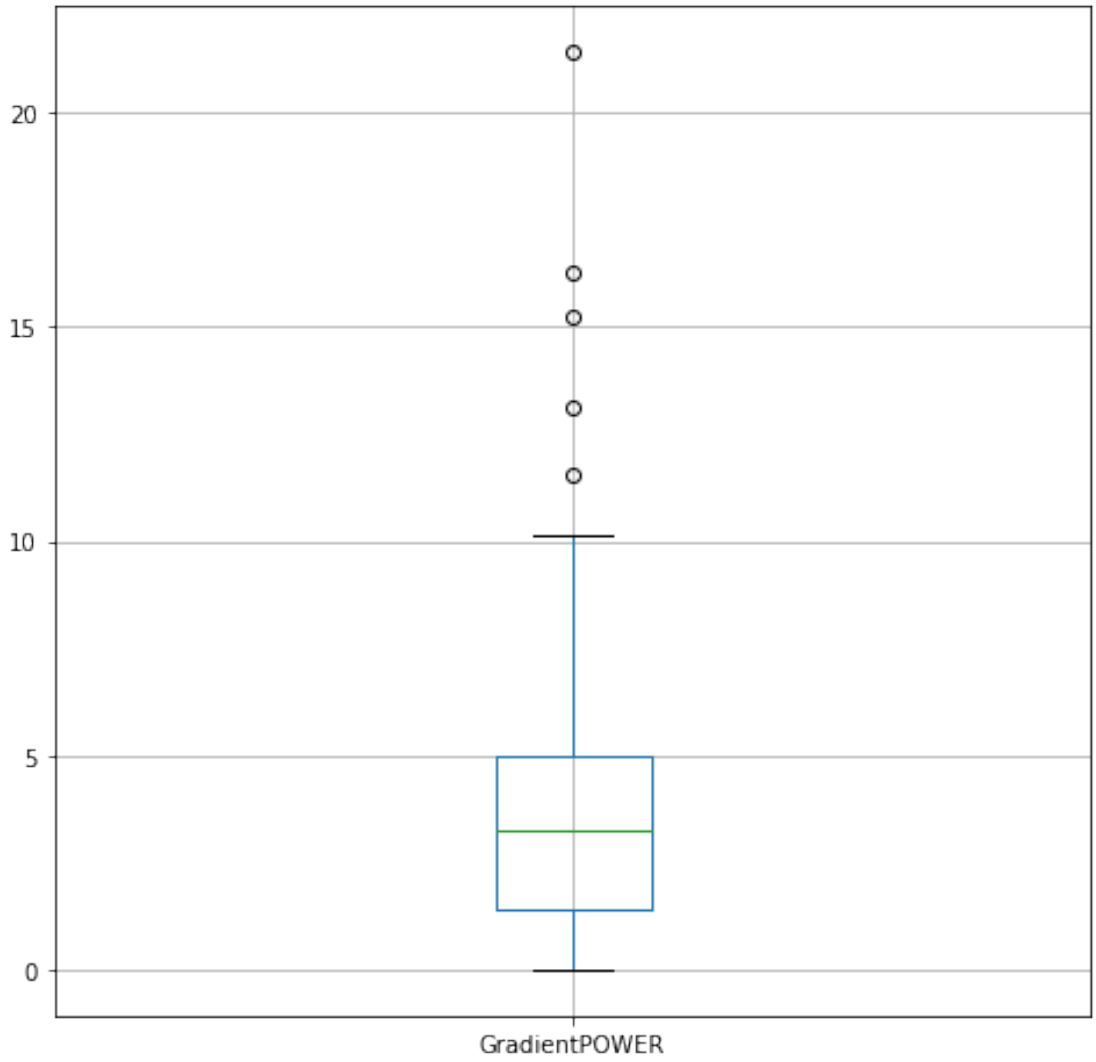


En detalle VelocityPOWER y GradientPOWER:

```
In [82]: Zones.boxplot(column='VelocityPOWER',figsize=(8,8));
```



In [84]: Zones.boxplot(column='GradientPOWER', figsize=(8,8));

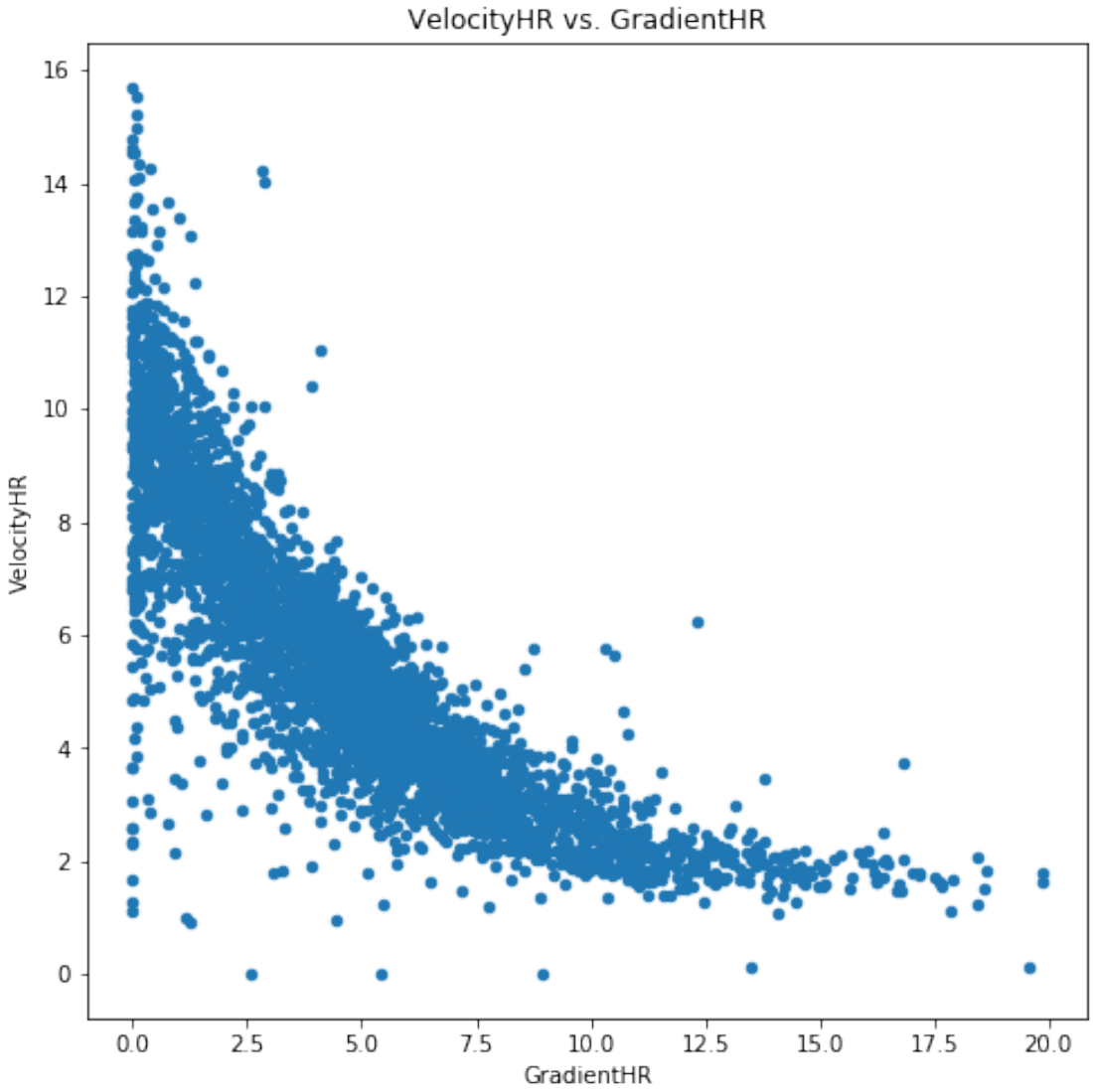


En la siguiente sección, se intentan relacionar diferentes parámetros.

1.3 Relación entre parámetros

```
In [103]: Zones_new=Zones[Zones.GradientHR<20][Zones.VelocityHR<20]
          Zones_new.plot.scatter(y='VelocityHR',
                                 x='GradientHR',
                                 figsize=(8,8),
                                 title='VelocityHR vs. GradientHR');
```

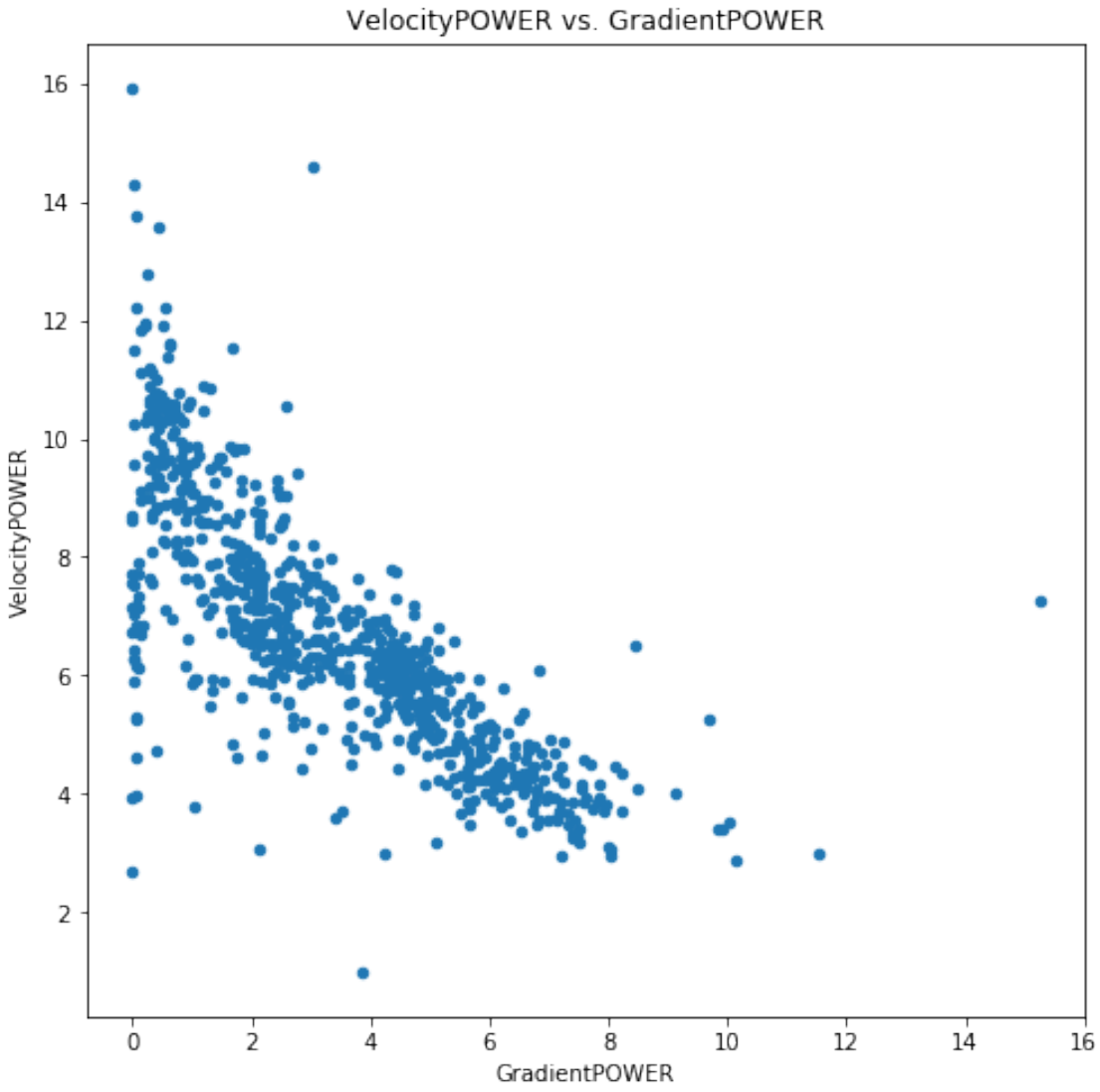
```
/usr/lib/python3/dist-packages/ipykernel_launcher.py:1: UserWarning: Boolean Series key will be
  """Entry point for launching an IPython kernel.
```



Como se observa existe una relación entre la velocidad y la pendiente que sigue una asímptota con límite 2 m/s. Además hay algunos puntos que denotan valores de lectura anormales y otros debidos a un bajo esfuerzo del ciclista.

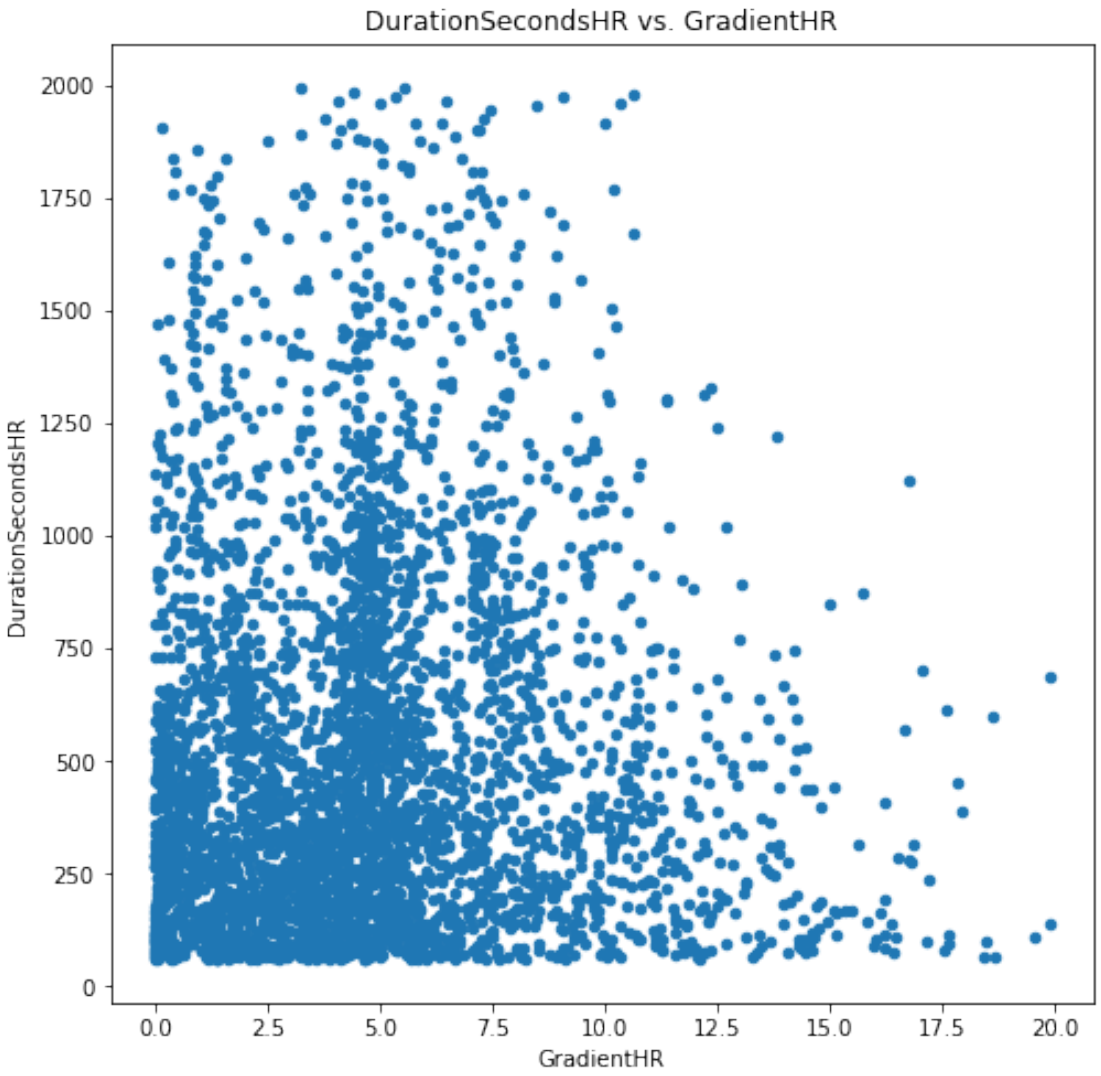
```
In [104]: Zones_new=Zones_new[Zones.GradientPOWER<20][Zones.VelocityPOWER<20]
          Zones_new.plot.scatter(y='VelocityPOWER',
                                 x='GradientPOWER',
                                 figsize=(8,8),
                                 title='VelocityPOWER vs. GradientPOWER');
```

```
/usr/lib/python3/dist-packages/ipykernel_launcher.py:1: UserWarning: Boolean Series key will be
  """Entry point for launching an IPython kernel.
```




```
In [13]: Zones_new_2=Zones[Zones.GradientHR<20][Zones.DurationSecondsHR<2000]
         Zones_new_2.plot.scatter(y='DurationSecondsHR',
                                  x='GradientHR',
                                  figsize=(8,8),
                                  title='DurationSecondsHR vs. GradientHR');
```

```
/usr/lib/python3/dist-packages/ipykernel_launcher.py:1: UserWarning: Boolean Series key will be
  """Entry point for launching an IPython kernel.
```



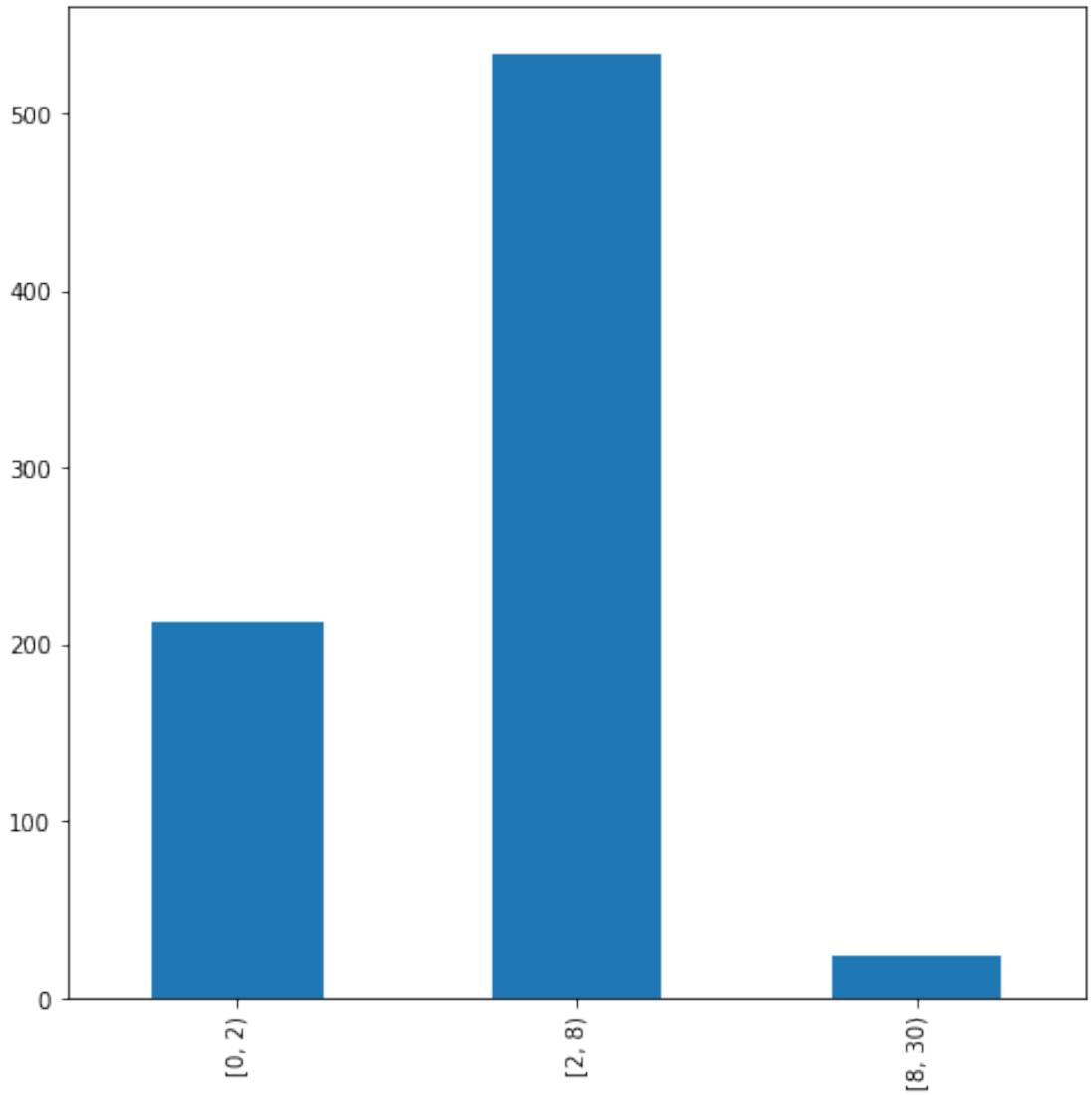
A diferencia del caso anterior, aparentemente no existe ninguna correlación entre la pendiente y el parámetro resistencia.

```
In [120]: (Zones_new.GradientHR<0).value_counts()

Out[120]: False    771
          Name: GradientHR, dtype: int64
```

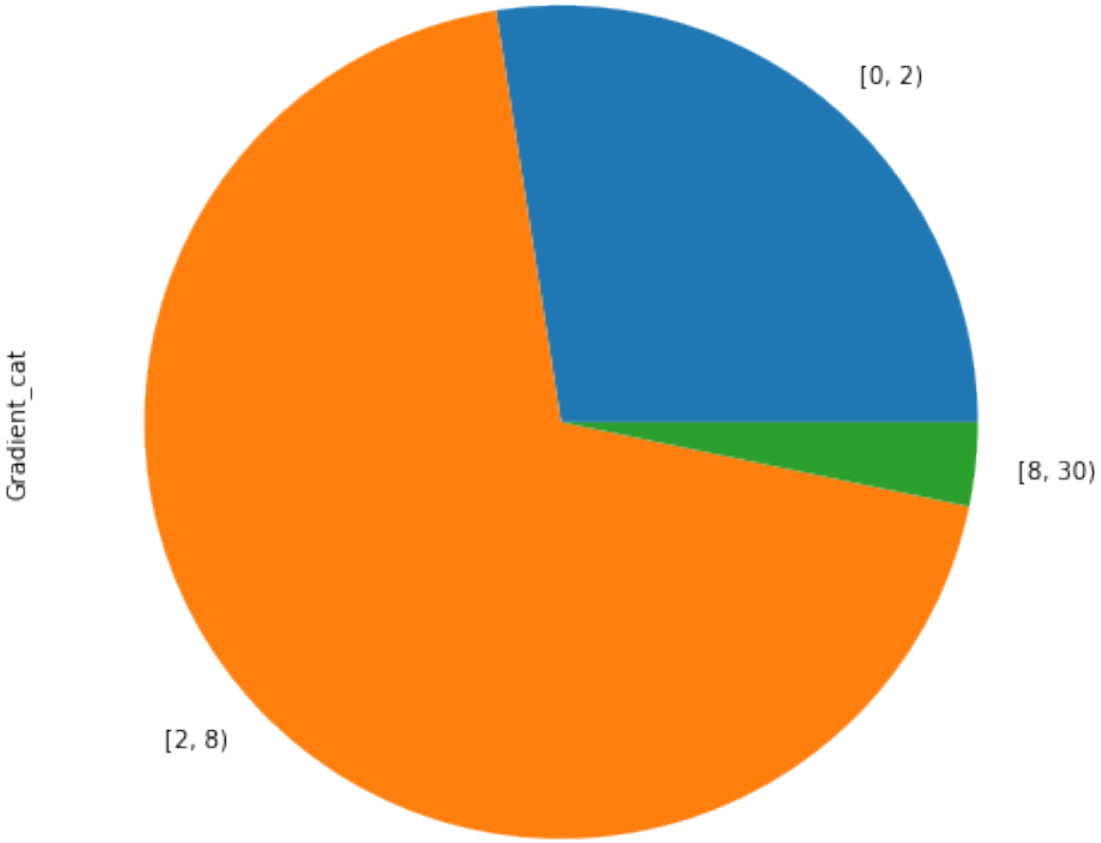
Ahora se observa más en detalle la pendiente y se procede a categorizar la variable. Se escoge HR porque hay muchos más registros que en POWER.

```
In [125]: bins = pd.IntervalIndex.from_tuples([(0, 2), (2, 8), (8, 30)], closed='left')
          bins

Out[125]: IntervalIndex([[0, 2), [2, 8), [8, 30)],
                        closed='left',
                        dtype='interval[int64]')
```

```
In [126]: pd.cut(Zones_new.GradientHR,
                 bins)

Out[126]: 353      [2, 8)
          354      [2, 8)
          356      [2, 8)
          357      [0, 2)
          359      [2, 8)
                   ...
          4291     [2, 8)
          4294     [2, 8)
          4297    [8, 30)
          4300     [2, 8)
          4302     [0, 2)
          Name: GradientHR, Length: 771, dtype: category
          Categories (3, interval[int64]): [[0, 2) < [2, 8) < [8, 30)]

In [127]: Gradient_cut = pd.cut(Zones_new.GradientHR,
                                bins);

In [128]: Zones_new['Gradient_cat'] = Gradient_cut.astype('str')

In [129]: Zones_new.Gradient_cat.value_counts().sort_index()

Out[129]: [0, 2)     212
          [2, 8)     534
          [8, 30)     25
          Name: Gradient_cat, dtype: int64

In [130]: Zones_new.Gradient_cat.value_counts().sort_index().plot.bar(figsize=(8,8));
```




```
In [131]: Zones_new.Gradient_cat.value_counts().sort_index().plot.pie(figsize=(8,8));
```



Transformando la pendiente en una variable categorica que se puede traducir como pendiente del 0%-2%, 2%-8% y >8% casi el 75% de los valores corresponden al intervalo 2%-8%. Sabiendo esta información, se puede descartar parámetros obtenidos en pendientes 0%-2% y >8% a modo de filtrado o por otro lado, usarlas exclusivamente para observar el rendimiento del ciclista en montaña o llano.

Anexo 3. Análisis estadístico de atletismo

PreprocessingRunning

October 22, 2020

1 Análisis estadístico del deporte RUNNING

In [1]:
```
# Actualizar packages
!pip3 install pandas --user --upgrade --quiet
!pip3 install numpy --user --upgrade --quiet
!pip3 install scipy --user --upgrade --quiet
!pip3 install statsmodels --user --upgrade --quiet
%load_ext autoreload
```

In [1]:
```
#%matplotlib notebook
%reload_ext autoreload
import numpy as np
import matplotlib.pyplot as plt
#import seaborn as sn
import pandas as pd
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
pd.set_option('precision', 3)
```

In [2]:
```
# extra imports
from pandas import read_csv
#from sklearn.neighbors import KNeighborsClassifier
from statsmodels.genmod.generalized_linear_model import GLM
from pandas.plotting import scatter_matrix
from scipy.stats import boxcox, shapiro
from statsmodels.graphics.gofplots import qqplot
```

In [3]:
```
Zones = read_csv("dataRUNNING.csv", header=0, delimiter=',')
Zones.shape
```

Out[3]:
```
(575, 4)
```

In [6]:
```
Zones.columns
```

Out[6]:
```
Index(['Date', 'DurationSecondsHR', 'VelocityHR', 'GradientHR'], dtype='object')
```

In [7]:
```
Zones[:4]
```

```
Out[7]:    Date  DurationSecondsHR  VelocityHR  GradientHR
        0   769               1057       2.843       1.060
        1   762                309       2.487       4.582
        2   749               1311       3.215       0.826
        3   747                481       2.638       1.533
```

1.1 Inspección básica de los datos

```
In [8]: Zones.describe()
```

```
Out[8]:            Date  DurationSecondsHR  VelocityHR  GradientHR
        count   575.000            575.000     575.000     370.000
        mean   1440.376            650.019       3.110       2.810
        std     477.563            719.589       0.792       4.167
        min       7.000             60.000       0.242       0.000
        25%    1414.000            187.000       2.728       0.824
        50%    1686.000            391.000       3.144       1.554
        75%    1722.500            841.500       3.521       3.288
        max    1792.000           5605.000       7.618      48.223
```

DurationSecondsHR tiene un mínimo de 60 segundos exactos porque es donde se situa el corte para considerar la zona de esfuerzo de un parámetro o no.

Los máximos de las variables DurationSecondsHR y GradientHR seguramente son outliers, aunque cabe destacar que el máximo de DurationSecondsHR puede ser por una actividad muy larga como una media maratón.

A continuación se examinan cuántos valores son null.

```
In [10]: (Zones.Date.isnull()).value_counts()
         (Zones.DurationSecondsHR.isnull()).value_counts()
         (Zones.VelocityHR.isnull()).value_counts()
         (Zones.GradientHR.isnull()).value_counts()
```

```
Out[10]: False    575
         Name: Date, dtype: int64
```

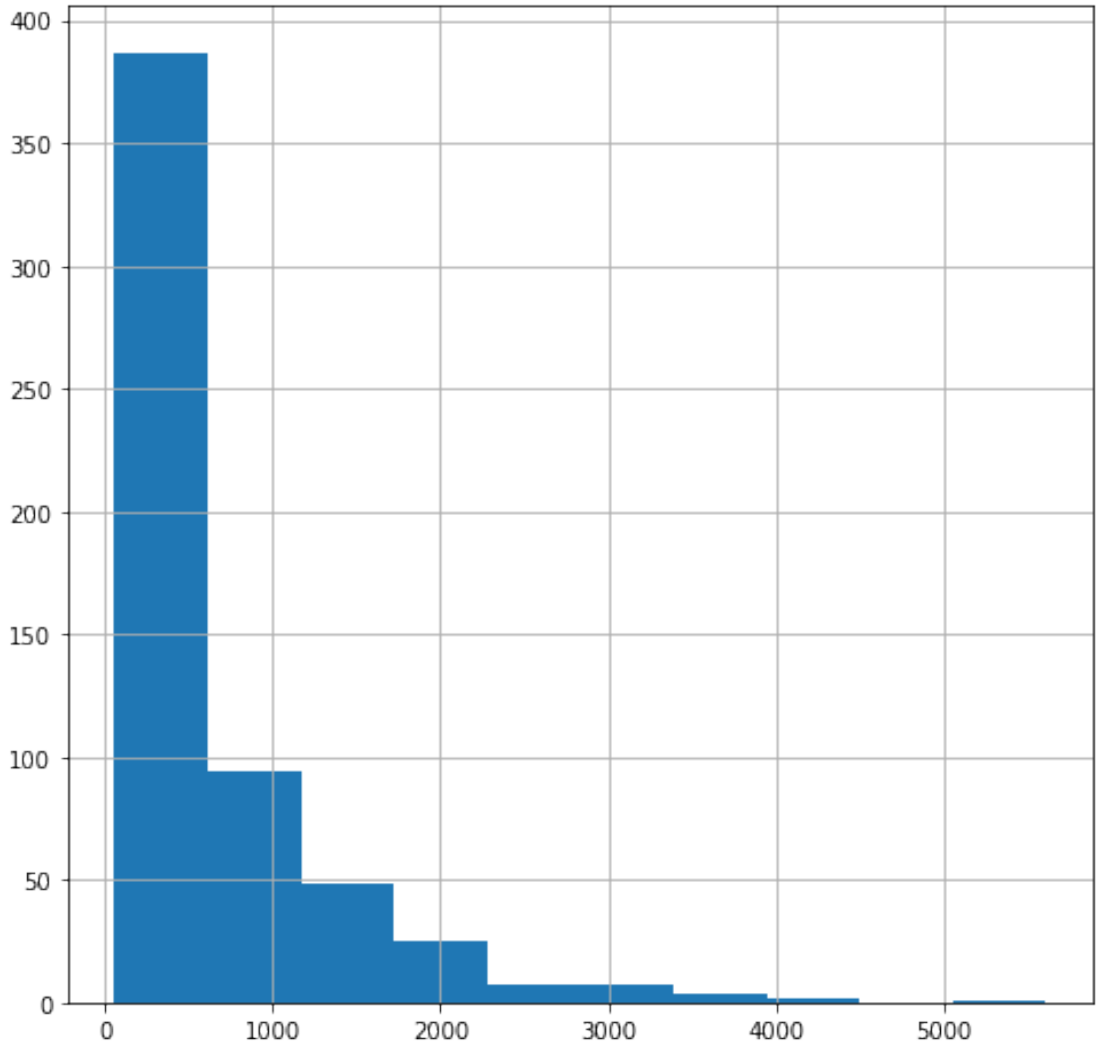
```
Out[10]: False    575
         Name: DurationSecondsHR, dtype: int64
```

```
Out[10]: False    575
         Name: VelocityHR, dtype: int64
```

```
Out[10]: False    370
         True     205
         Name: GradientHR, dtype: int64
```

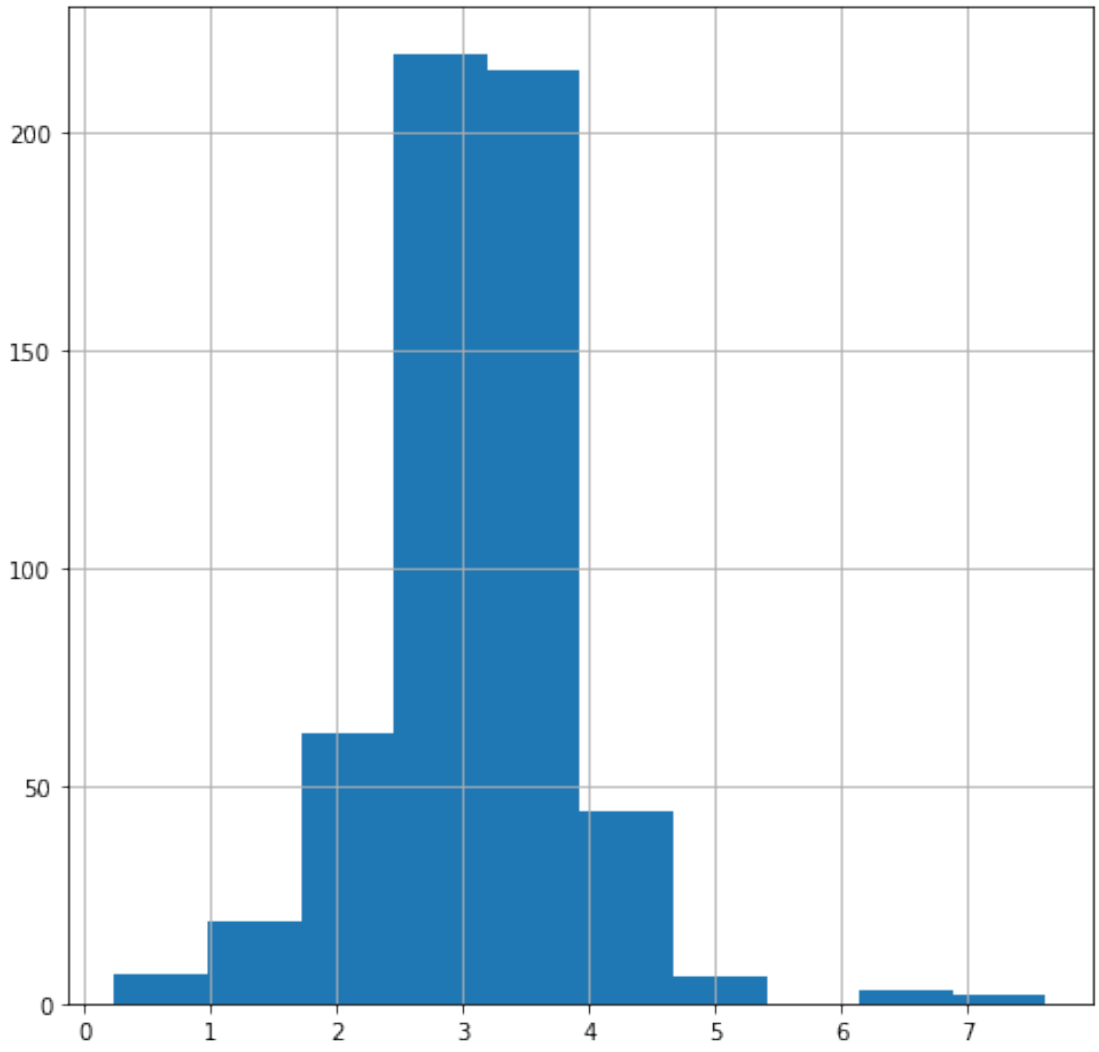
No hay ningún valor null por lo tanto los datos estan bastante limpios exceptuando el caso de GradientHR. Esto se debe a que en la actividad que ha registrado el atleta no hay la información adecuada en cuanto a elevación o altitud para calcular la pendiente y por lo tanto no se puede realizar el cálculo.

```
In [11]: Zones.DurationSecondsHR.hist(figsize=(8,8));
```



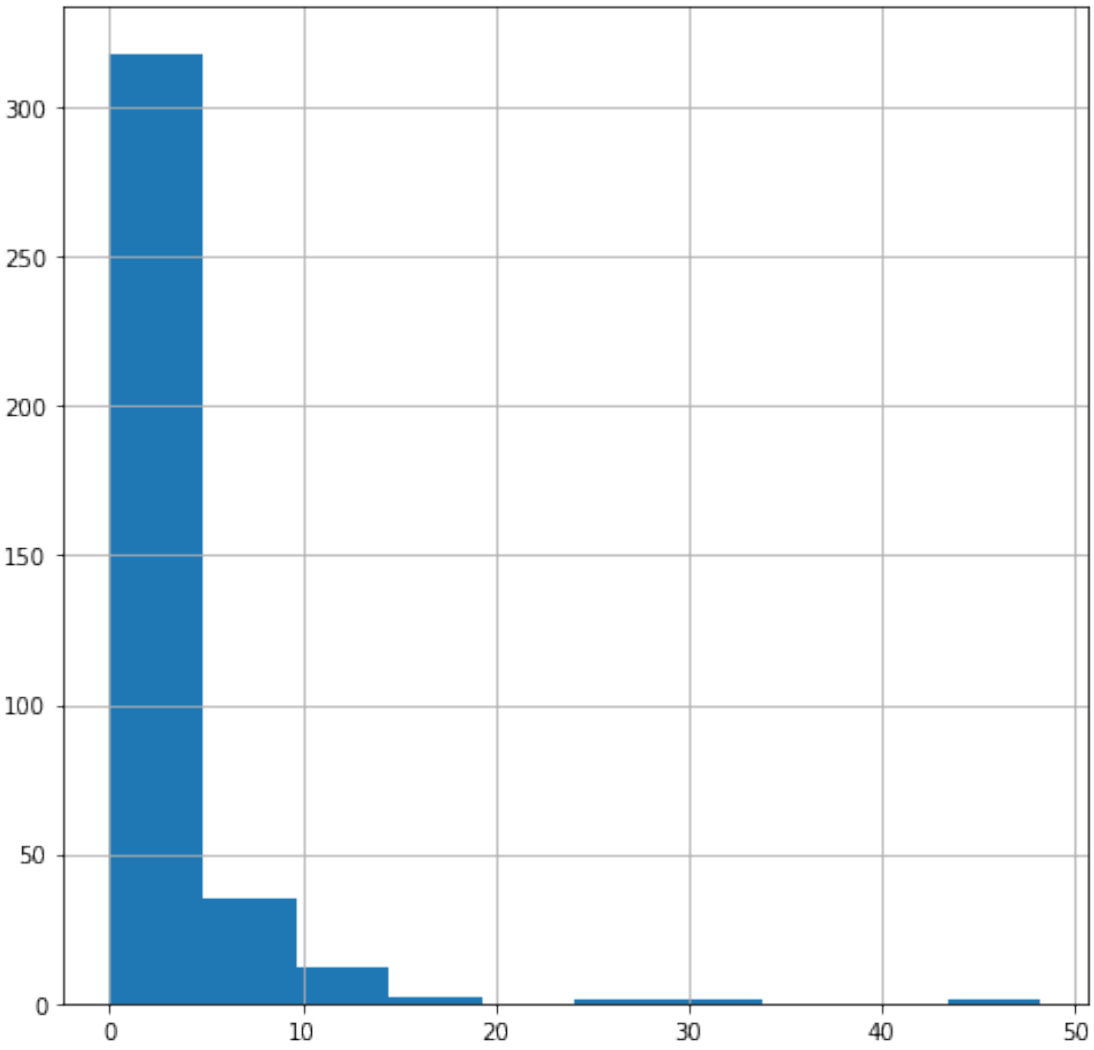
Por lo explicado anteriormente, no queda claro si la barra por encima de 5000 es un outlier.
Histograma VelocityHR:

```
In [14]: Zones.VelocityHR.hist(figsize=(8,8));
```



No se observan outliers a simple vista.

```
In [20]: Zones.GradientHR.hist(figsize=(8,8));
```

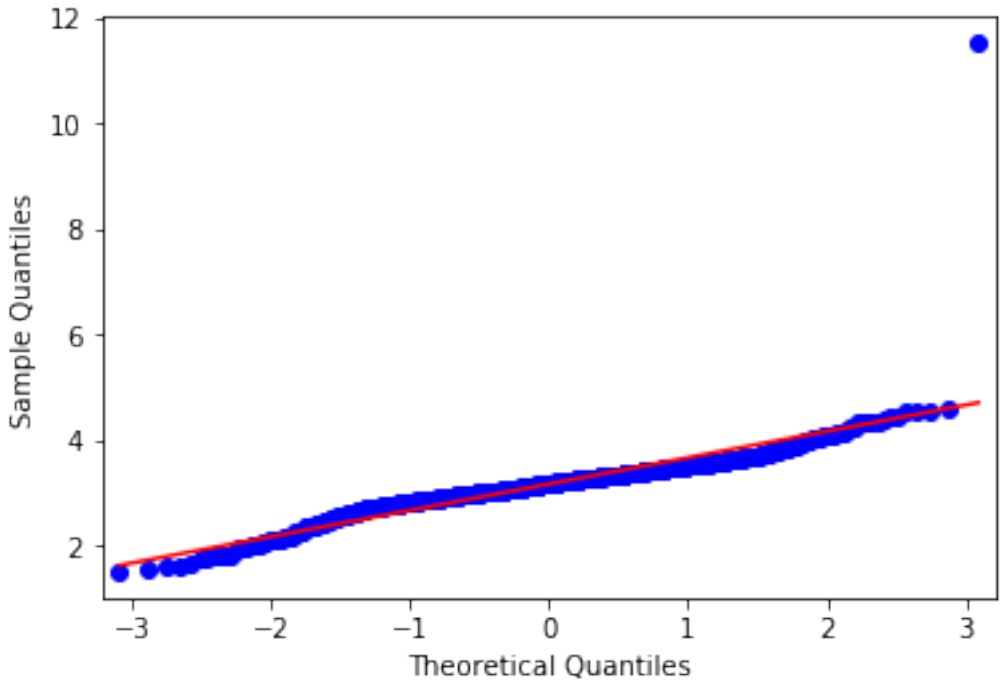


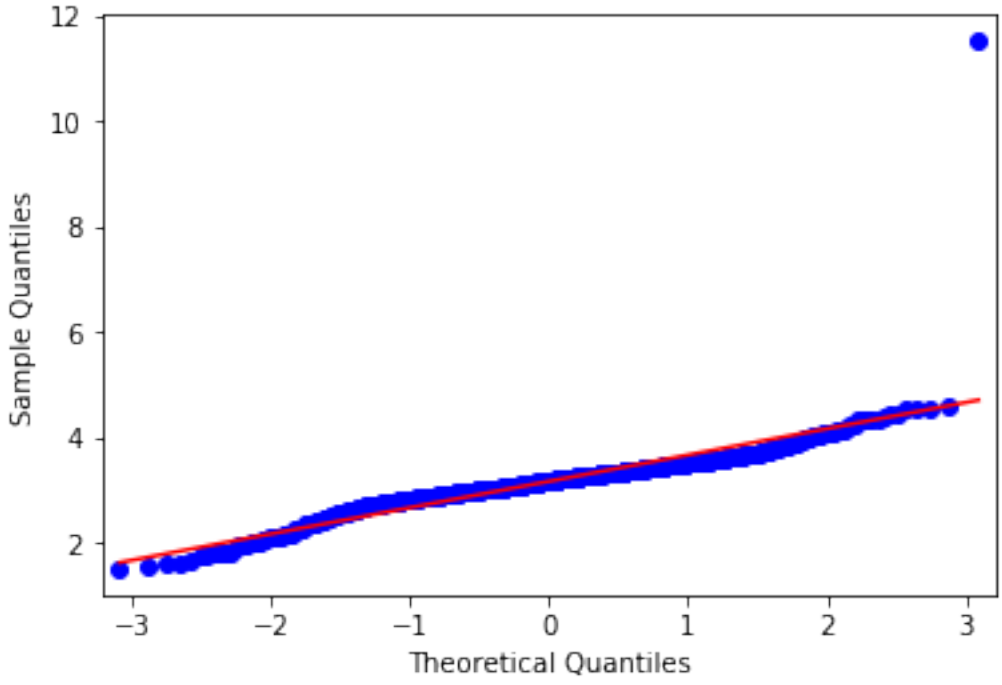
En este parámetro los valores por encima de 20 queda claro que son outliers.

Cabe destacar que por la forma de las gráficas, DurationSecondsHR sigue una distribución más parecida a una Poisson con = 1. Por otro lado, VelocityHR tiene una forma más Gaussiana, por lo tanto se ejecuta un test de normalidad:

```
In [30]: data = Zones.VelocityHR.dropna()
         qqplot(data, line='s')
         plt.show()
```

Out[30]:





Parace que salvo que un dato que debe ser un outlier que no se observa con claridad en el histograma, los datos se ajustan a la diagonal de una distribución Gaussiana.

```
In [33]: shapiro_test = shapiro(data)
         shapiro_test

Out[33]: ShapiroResult(statistic=0.7835566997528076, pvalue=2.6750862694907034e-34)
```

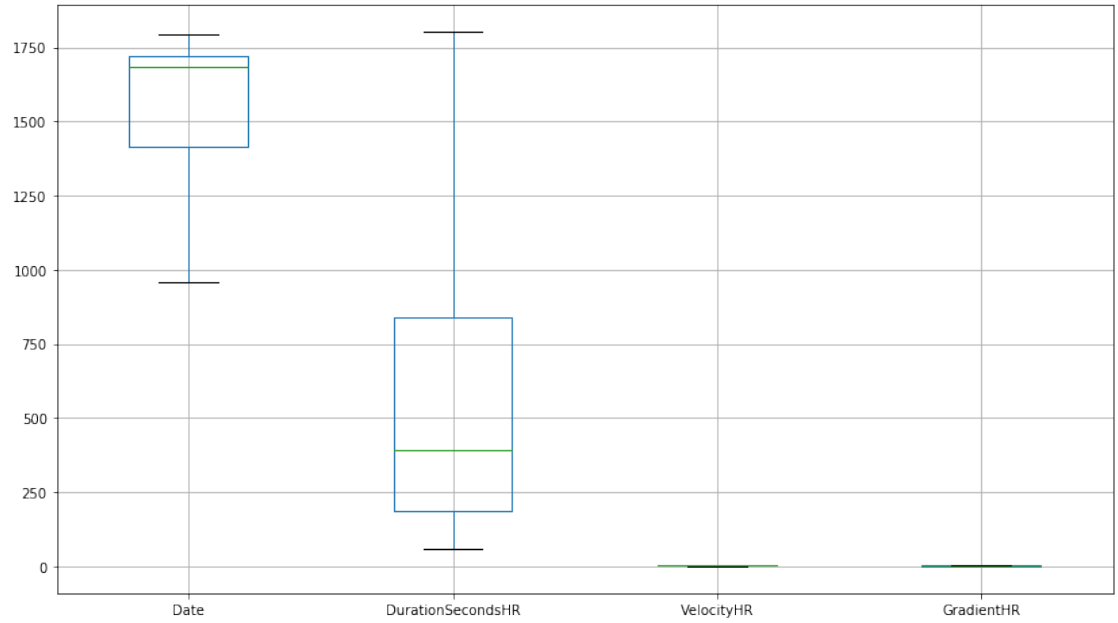
A pesar de los gráficos, el test da un p-value muy pequeño que indica que los datos no forman una distribución Gaussiana. Por lo tanto, se concluye que los datos de este parámetro solamente siguen una distribución unimodal.

1.2 Resumen gráfico

En este deporte solo hay parámetros HR, ya que POWER está relacionado al ciclismo únicamente.

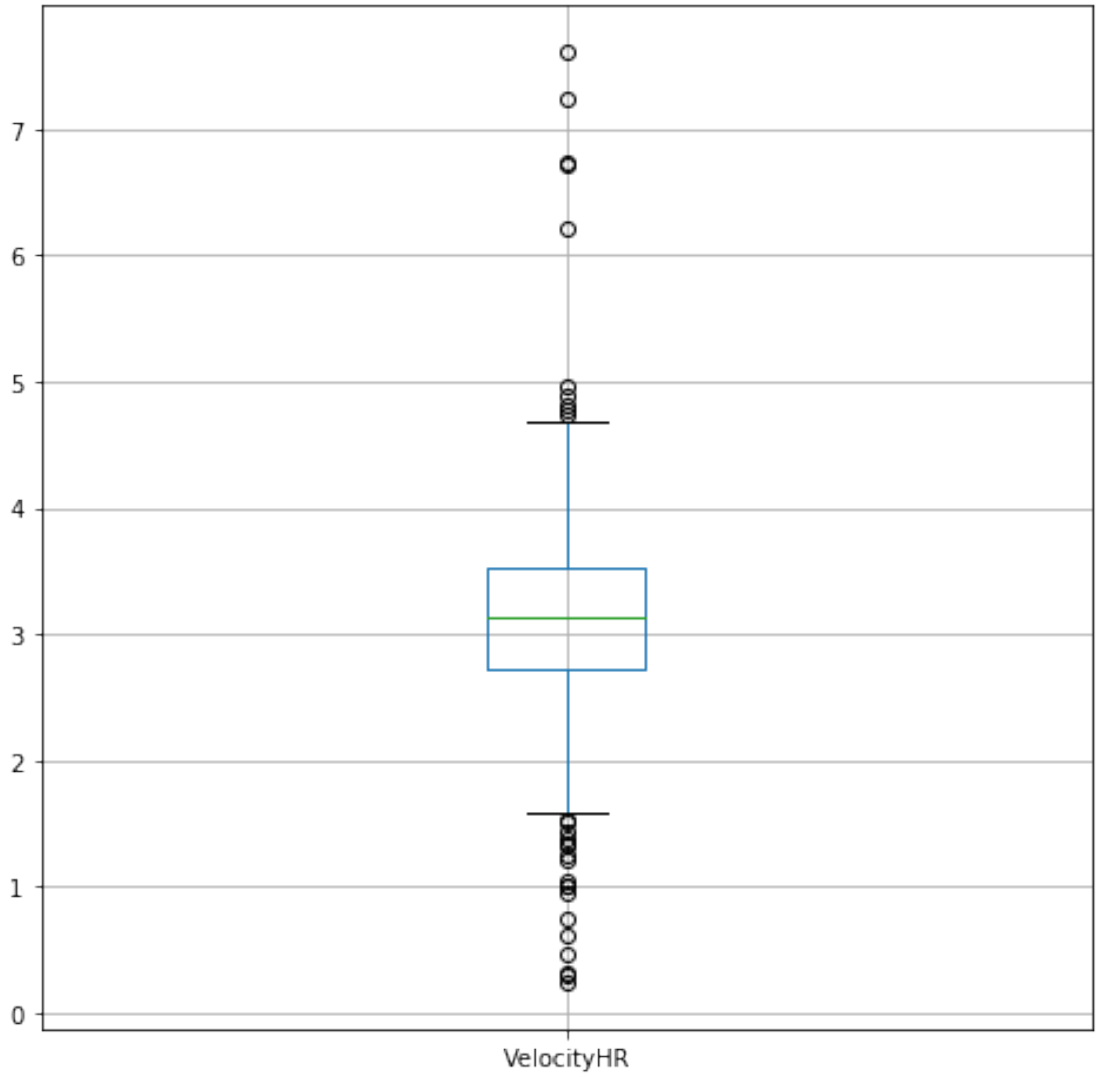
```
In [17]: Zones.loc[:,"Date":"GradientHR"].boxplot(figsize=(14,8), showfliers=False);
```

```
/home/david/.local/lib/python3.6/site-packages/numpy/core/_asarray.py:83: VisibleDeprecationWa
  return array(a, dtype, copy=False, order=order)
```

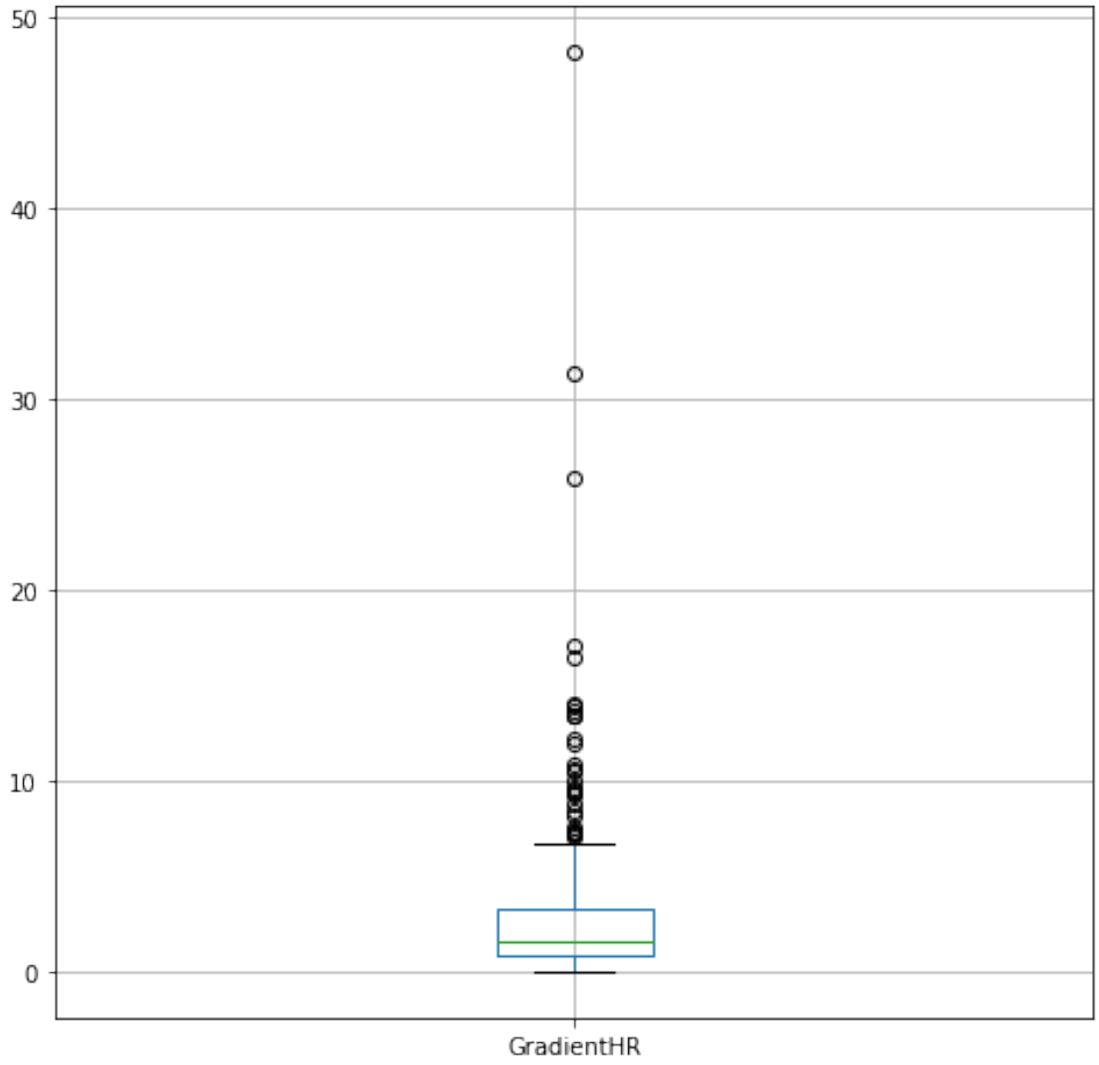


En detalle VelocityHR y GradientHR:

```
In [18]: Zones.boxplot(column='VelocityHR',figsize=(8,8));
```

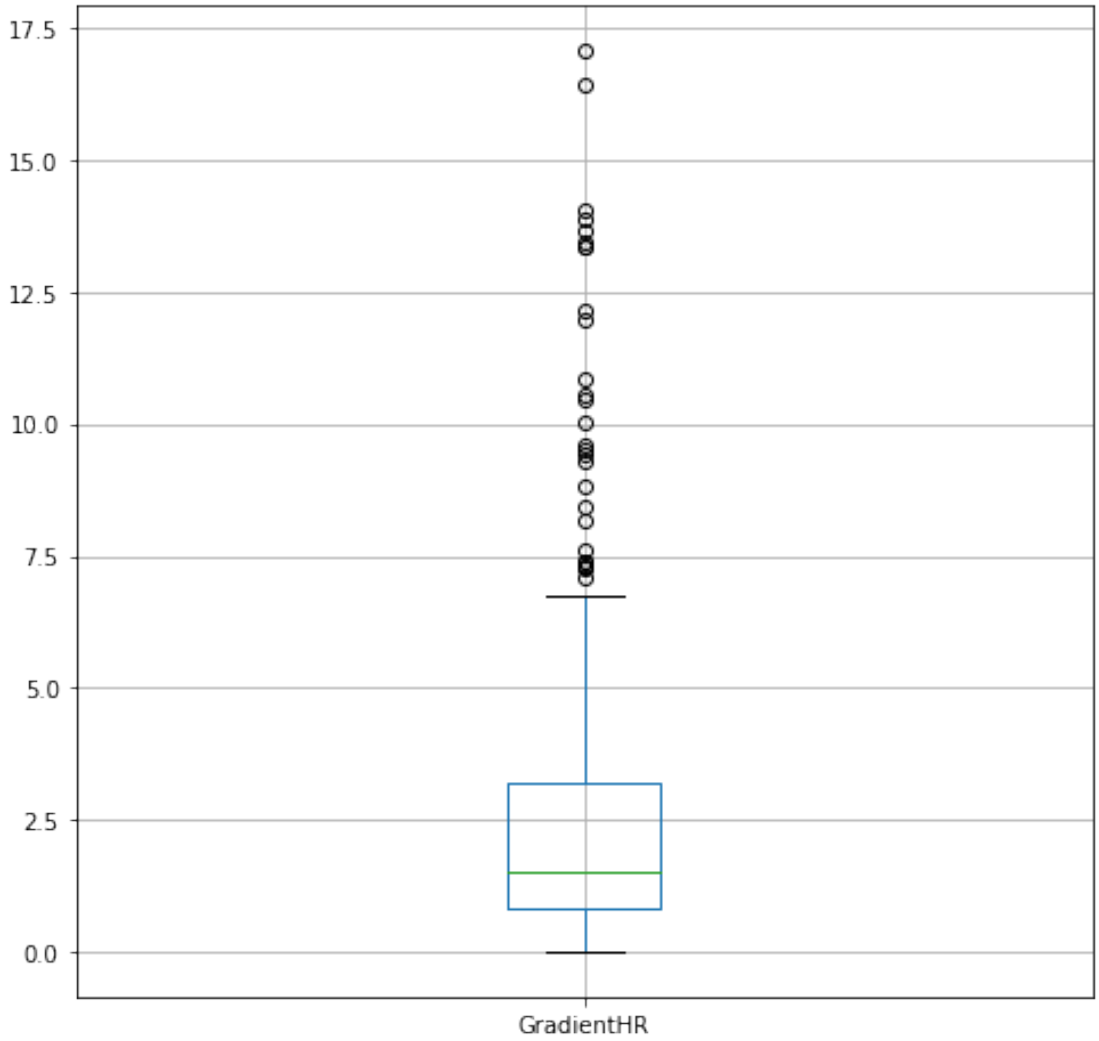



```
In [19]: Zones.boxplot(column='GradientHR',figsize=(8,8));
```



Limpiando outliers:

```
In [22]: Zones[Zones.GradientHR<20].boxplot(column='GradientHR',figsize=(8,8));
```

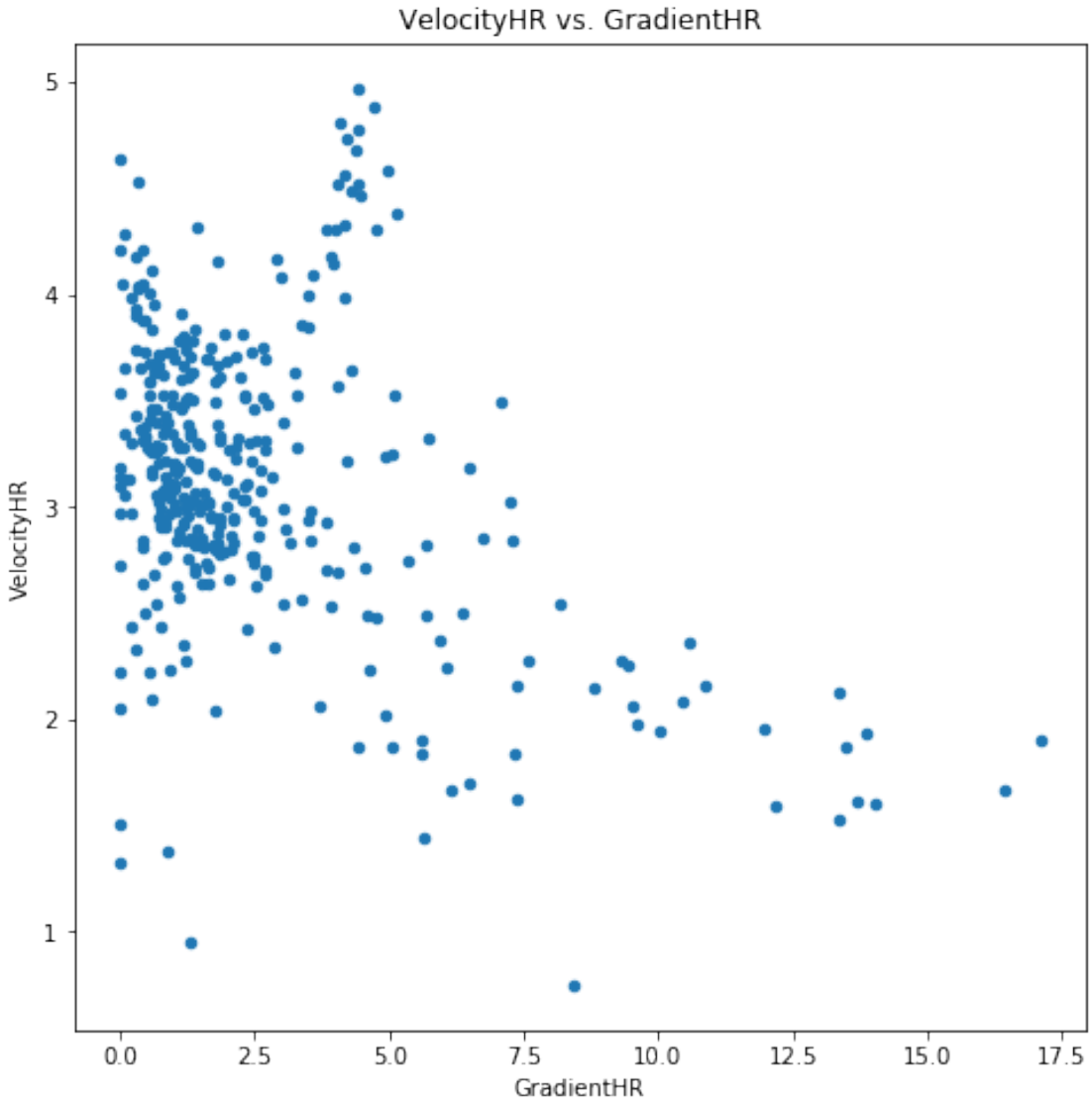


Ahora se puede ver con más exactitud que la mediana de los valores está por debajo de un 2% de pendiente y por la tanto hay muchas actividades que son en terrenos llanos. Además los círculos sueltos que estan por encima del boxplot remarcan un conjunto de actividades que han tenido secciones de pendiente muy elevada ya sean en montaña o en escaleras.

1.3 Relación entre parámetros

Relación entre VelocityHR y GradientHR:

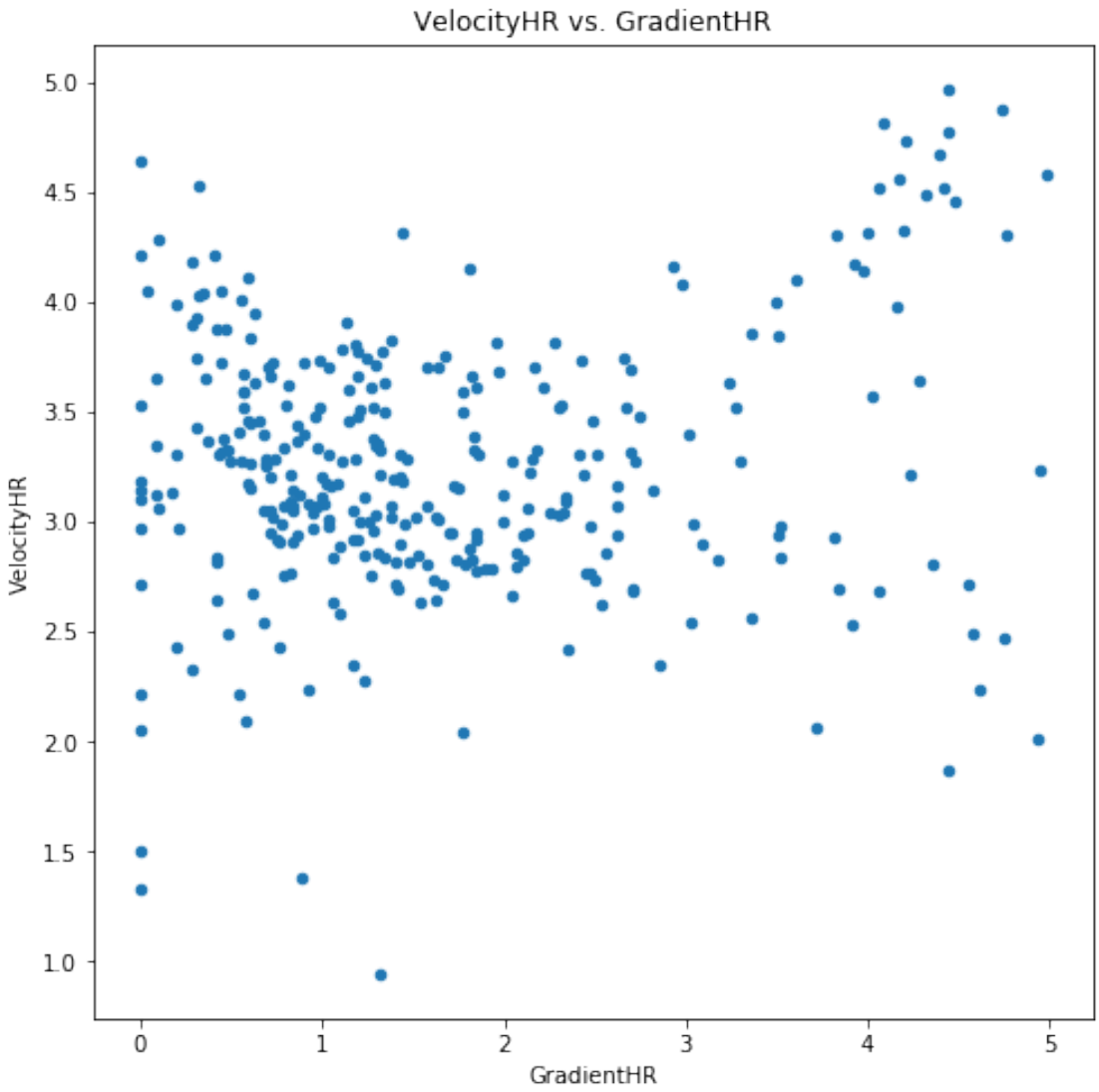
```
In [23]: Zones_new=Zones[Zones.GradientHR<20]
         Zones_new.plot.scatter(y='VelocityHR',
                                x='GradientHR',
                                figsize=(8,8),
                                title='VelocityHR vs. GradientHR');
```



En este caso y a diferencia de CYCLING_STREET, hay muchos menos datos y cuesta más ver una correlación clara. Aun así se ve una ligera curva en las actividades con pendiente entre 5 y 15 pero con unos valores de velocidad muy altos antes de la pendiente 5 que no siguen el patrón. Aun así, todas las actividades con más de 5% de pendiente podrían apartarse del resto sin ningún problema por su escasez y ya que en conjunto tienen valores de velocidad menores.

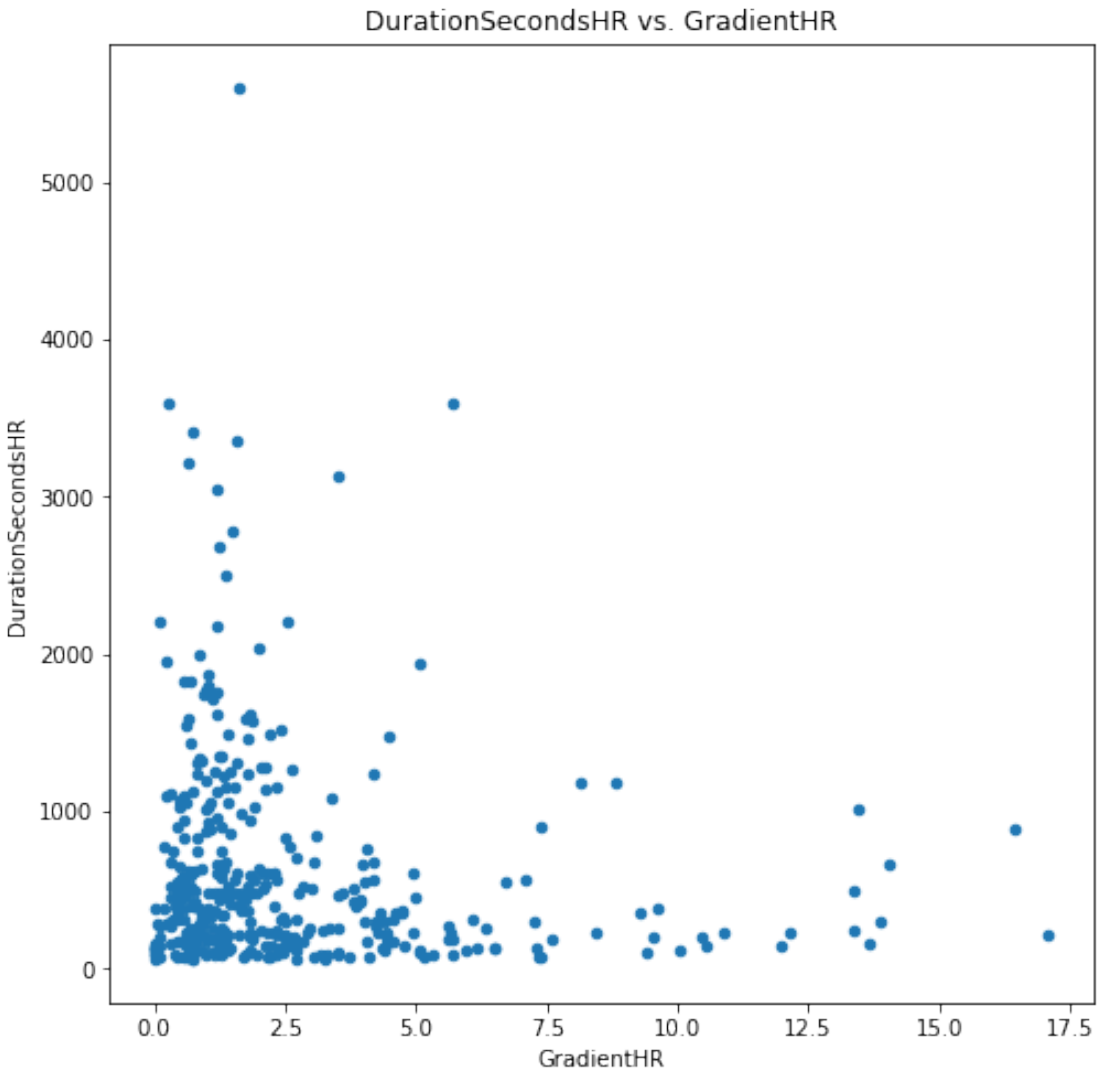
En más detalle:

```
In [25]: Zones_new=Zones[Zones.GradientHR<5]
         Zones_new.plot.scatter(y='VelocityHR',
                                x='GradientHR',
                                figsize=(8,8),
                                title='VelocityHR vs. GradientHR');
```



Se observa que si que hay una leve tendencia hacia abajo en velocidad cuando aumenta la pendiente, pero no es tan clara como sucede en ciclismo. Por lo tanto, no sería muy conveniente hacer una clasificación por pendiente para luego elaborar un filtro.

```
In [26]: Zones_new_2=Zones[Zones.GradientHR<20]
         Zones_new_2.plot.scatter(y='DurationSecondsHR',
                                  x='GradientHR',
                                  figsize=(8,8),
                                  title='DurationSecondsHR vs. GradientHR');
```



La gráfica muestra de forma clara que no existe ningún tipo de relación entre DurationSecondsHR y GradientHR.