

EPHEC

Avenue Konrad Adenauer 3
1200 Bruxelles

DEBONGNIE Nathan
BDA00350
3BDAA

Analyse et prédition de l'évolution des performances des nageurs Belges

Travail de fin d'études présenté
pour l'obtention du diplôme de
bachelier de spécialisation en Business Data Analysis

Année académique 2024-2025

I. Avant-propos

Remerciements

Avant toute chose, je souhaite remercier les personnes qui ont rendu la réalisation de ce travail possible.

Premièrement, je tiens à exprimer ma gratitude à mes proches et amis, dont le soutien moral m'a été précieux tout au long de mes études et de ce travail. Un remerciement particulier à Matthieu Luyckx, Data Scientist chez Jetpack.AI, pour les conseils et son expertise en data mining.

Je remercie également Christian Kaufmann (results@swimrankings.net) et Swimrankings en général pour les autorisations et l'accessibilité des données utilisées dans le cadre de ce projet.

Déclaration de non plagiat

Année académique 2024-2025

Haute Ecole **EPHEC**

BACHELIER DE
SPECIALISATION EN
BUSINESS DATA
ANALYSIS

Je soussigné(e) *Nathan DEBONGNIE* ,

Étudiant(e) de la 3BDAA (Bachelier spécialisé en Business Data Analysis) , déclare par la présente que le travail ci- joint est exempt de tout plagiat et respecte en tous points le règlement des études en matière d'emprunts, de citations et d'exploitation de sources diverses signé lors de mon inscription à l'EPHEC, ainsi que les instructions et consignes mises à ma disposition sur le Moodle et/ou l'intranet étudiants. Par ma signature, je certifie sur l'honneur avoir pris connaissance des documents précités et que le travail présenté est original et exempt de tout emprunt à un tiers non cité correctement.

Date : 29-05-25

Signature :

DEBONGNIE Nathan

Déclaration de l'usage de l'IA générative dans le cadre du travail de fin d'études

L'encart suivant doit être complété par l'étudiant et inséré sur la première page de son travail écrit, juste après la page de garde son travail de fin d'études. Attention, il est possible de cocher plusieurs cases.

Tout étudiant qui réalise un travail écrit doit remplir l'encart relatif à l'utilisation de l'IA générative au même titre que la déclaration de plagiat.

Le but n'est pas de dissuader les étudiants de remplir cet encart, c'est-à-dire que si l'étudiant joue le jeu et que son utilisation de l'IA générative est intelligente, ce dernier ne doit pas être sanctionné pour la seule raison d'avoir utilisé une IA générative.

Néanmoins, si un étudiant fait une déclaration mensongère sur son utilisation, par exemple : "j'ai écrit l'intégralité de mon texte sans avoir eu recours à un outil d'IA générative" alors qu'il s'avère qu'il l'a utilisé, celui-ci s'expose à des sanctions. Lors de sa défense orale, l'étudiant pourra être interrogé de manière plus approfondie sur les parties de textes qui semblent poser question. L'étudiant pourrait être sanctionné dans l'hypothèse où les doutes concernant la fausse déclaration seraient confirmés par un examen approfondi.

Dans cette étude, le rôle de l'IA générative a été :

- | |
|--|
| <input type="checkbox"/> A) J'ai écrit l'intégralité de mon texte sans avoir eu recours à un outil d'IA générative ; |
| <input type="checkbox"/> B) J'ai rédigé le contenu de mon travail mais j'ai sollicité un outil d'IA générative pour améliorer |
| <input type="checkbox"/> C) L'orthographe |
| <input type="checkbox"/> D) La grammaire |
| <input type="checkbox"/> E) La syntaxe |
| <input checked="" type="checkbox"/> F) C) J'ai consulté un outil d'IA générative pour m'inspirer et puiser des idées de rédaction au niveau du contenu ou de la structure ; |
| <input checked="" type="checkbox"/> G) D) J'ai construit des idées que j'ai ensuite soumises à un outil d'IA générative qui m'a aidé à développer mon texte sur base de ces idées ; |
| <input type="checkbox"/> H) E) J'ai sollicité un outil d'IA générative à des fins de traduction ; |
| <input type="checkbox"/> I) F) J'ai confronté plusieurs propositions de contenu produit par l'IA générative pour en sélectionner les passages les plus pertinents, et j'ai édité et amélioré pour la plupart ; |
| <input type="checkbox"/> J) G) J'ai édité et amélioré une proposition de contenu produit par l'IA générative ; |
| <input type="checkbox"/> K) H) Une ou des parties de mon travail ont été intégralement produites au moyen d'un outil d'IA générative sans apport original de ma part. |

Ci-dessous, il est indiqué ce qui est attendu en matière de référencement, sur base des usages des outils d'IA générative renseignés ci-dessus. Les différentes mentions devront être rassemblées dans une section dédiée (intitulée « Mentions des usages, prompts et productions d'outils d'IA générative dans le cadre de ce travail ») en fin d'étude.

A) J'ai écrit l'intégralité de mon texte sans avoir eu recours à un outil d'IA générative ;

⇒ Aucune démarche de référencement de l'outil d'IA générative n'est requise.

B) J'ai rédigé le contenu de mon travail mais j'ai sollicité un outil d'IA générative pour améliorer :

- l'orthographe,
- la grammaire,
- Et/ou la syntaxe ;

⇒ Cet usage d'outil d'IA générative ne doit faire l'objet d'aucune démarche de référencement académique.

C) J'ai consulté un outil d'IA générative pour m'inspirer et puiser des idées de rédaction au niveau du contenu ou de la structure ;

⇒ Cet usage nécessite de mentionner le ou les outils d'IA générative utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
⇒ En outre, l'étudiant tiendra à disposition des évaluateurs le relevé des échanges avec l'outil d'IA générative et devra les produire en cas de demande explicite.

D) J'ai construit des idées que j'ai ensuite soumises à un outil d'IA générative qui m'a aidé à développer mon texte sur base de ces idées ;

⇒ Cet usage nécessite de mentionner le ou les outil(s) d'IA générative utilisé(s) en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
⇒ En outre, l'étudiant tiendra à disposition des évaluateurs le relevé des échanges avec l'outil d'IA générative et devra les produire en cas de demande explicite.

E) J'ai sollicité un outil d'IA générative pour traduire du contenu que j'ai produit dans ma langue maternelle vers une langue-cible ;

⇒ Cet usage nécessite de mentionner :

- le ou les outil(s) d'IA générative utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
- la ou les parties traduites ayant fait l'objet d'une traduction.

F) J'ai confronté plusieurs propositions de contenu produits par l'IA générative pour en sélectionner les passages les plus pertinents, que j'ai édité et amélioré pour la plupart ;

⇒ Cet usage nécessite de mentionner :

- le ou les outil(s) d'IA générative utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
- les requêtes (*prompts*) utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
- l'intégralité des propositions produites par l'outil d'IA générative dans un encart dédié à cet effet (voir exemple) après la bibliographie ;

G) J'ai édité et amélioré une proposition de contenu produit par l'IA générative ;

⇒ Cet usage nécessite de mentionner :

- i. le ou les outil(s) d'IA générative utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
- ii. les requêtes (*prompts*) utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
- iii. l'intégralité des propositions produites par l'outil d'IA générative dans un encart dédié à cet effet (voir exemple) après la bibliographie ;

H) Une ou des parties de mon travail ont été intégralement produites au moyen d'un outil d'IA générative sans apport original de ma part.

⇒ Cet usage nécessite de mentionner :

- i. le ou les outil(s) d'IA générative utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
- ii. les requêtes (*prompts*) utilisés en l'indiquant dans un encart dédié à cet effet (voir exemple) après la bibliographie ;
- iii. l'intégralité des propositions produites par l'outil d'IA générative dans un encart dédié à cet effet (voir exemple) après la bibliographie ;

II. Executive summary

Dans le monde du sport, la recherche de performance est un objectif constant. En natation, comprendre la progression des athlètes est essentiel pour optimiser l'évolution d'un ou une sportif.ve.

C'est sur cette logique que ce base l'objectif de ce travail. Les résultats de nombreuses compétitions sportives sont enregistrés, et il est possible d'en déduire un grand nombre d'informations permettant de prédire l'évolution future des performances individuelles.

Le projet est organisé en plusieurs étapes. La collecte automatisée des résultats via du web scraping, la structuration d'une base de données adaptée, l'exploration des données comportant une première phase d'analyse à l'aide de rapports graphiques (Power BI) et finalement le développement d'un modèle permettant de réaliser des prédictions de résultats associé à d'autres dashboards visuels qui complète les analyses.

A l'aide de cet outil, il deviendra plus aisé pour les entraîneurs et les clubs de natation d'adapter leurs programmes d'entraînement et d'identifier les nageurs à haut potentiel rapidement. Les nageurs eux-mêmes pourront profiter également de cet outil pour se projeter vers des objectifs réalisables à court ou moyen terme.

Table des matières

I.	Avant-propos.....	II
II.	Executive summary.....	VII
III.	Introduction.....	I
IV.	Méthode de travail.....	II
V.	Compréhension métiers	III
VI.	Récupération des données.....	IV
a.	Web scraping.....	V
b.	Base de données.....	VII
c.	Serveur.....	VIII
VII.	Compréhension des données.....	IX
a.	Visualisation	X
b.	Statistiques descriptives	XI
VIII.	Préparation des données	XII
a.	ETL.....	XII
b.	Feature engineering	XIV
IX.	Modélisation des données.....	XV
a.	Régression polynomiale	XVI
b.	Réseau de neurones	XVII
c.	Modèle final	XIX
d.	Prédictions.....	XXI
X.	Visualisation.....	XXII
a.	Clubs	XXII
b.	Nageurs.....	XXIII
XI.	Conclusion.....	XXIV
a.	Evaluation.....	XXIV
b.	Pistes d'amélioration	XXIV
c.	Difficultés rencontrées	XXV
XII.	Bibliographie.....	XXVI

III. Introduction

Depuis très jeune, le sport, et plus particulièrement la natation, occupe une place importante dans ma vie. Des entraînements presque quotidiens et des compétitions régulières toujours à la recherche de la meilleure performance possible. En parallèle de cela, je me suis découvert un attrait particulier pour les données. Ce travail de fin d'études est une opportunité idéale pour lier les 2.

La performance sportive est au cœur de toutes les disciplines sportives compétitives. Outre la réflexion sur les méthodologies d'entraînements, la nutrition ou l'équipement, certains sports intègrent une notion d'analyse de données pour améliorer les performances. Ce n'est pas encore vraiment le cas dans la natation, alors que beaucoup de données existent. Le site Swimrankings.net par exemple, reprends la majorité des résultats de natation dans divers pays. Ces informations pourraient-elles permettre de prévoir l'évolution d'un nageur ?

L'objectif principal de ce projet est donc de proposer un outil capable d'analyser les performances passées et de prédire l'évolution d'un nageur sur une course spécifique. Cet outil pourrait s'avérer utile pour tous les acteurs du sport de nage, que ce soit pour identifier les futurs talents, ou accompagner les nageurs vers des objectifs réalisables.

Pour mener à bien ce projet, il est organisé en plusieurs étapes. Une fois les objectifs clairement définis, la première étape est de comprendre le cadre et les besoins spécifiques au bon déroulement du projet. La partie suivante consiste à récupérer toutes les données utiles. Sur base de ces données et du sujet du projet, une analyse approfondie des données est nécessaire. Cette compréhension des données permet par la suite de réaliser les modifications nécessaires afin qu'elles soient propres et utilisables pour construire un modèle de prédictions.

Un des principaux défis du projet est la mise en pratique d'un modèle de data mining, d'autant plus dans un cadre aussi spécifique que celui-ci. En effet, il s'avère que faire de la prédiction sur des résultats sportifs n'est pas une tâche facile.

Ce document détaille les étapes ayant été nécessaires pour parvenir au résultat final du projet. En partant du constat qu'il devait être possible d'apporter une analyse constructive des données de résultats de compétitions, proposer un dashboard de visualisation pouvant aider les nageurs et les clubs pour comprendre et suivre les performances ainsi qu'un modèle prédictif de résultats pour pouvoir fixer des objectifs réalisables.

IV. Méthode de travail

Réaliser un projet aussi important nécessite une bonne organisation. Pour cela, je me suis basé sur la méthode « Crisp DM », qui permet de structurer l'approche d'un projet data. Il est composé de 6 phases bien définies, qui sont toutes aussi importantes que les autres. Le démarrage d'un projet data passe par une phase d'analyse et de réflexion. L'objectif est de déterminer la structure organisationnelle du projet, de définir le cadre et les objectifs SMART et de comprendre le contexte dans lequel le projet s'inscrit. La seconde partie est une continuité et est en lien rapproché avec la phase de compréhension métiers. La compréhension des données est une phase cruciale et souvent trop peu prise en compte à l'entame d'un projet. Elle permet d'étoffer sa compréhension du projet et d'identifier des informations clés supplémentaires. Récupérer des statistiques descriptives des variables et une première visualisation des valeurs sont les 2 techniques généralement mises en place. Il est parfois nécessaire de revenir à la compréhension métier pour s'assurer des informations extraites des données. Lorsque toutes les informations nécessaires au bon déroulement du projet sont récupérées, la partie technique peut commencer. Pour commencer, une phase de préparation des données doit être réalisée. Cette phase consiste à transformer toutes les données afin de pouvoir les utiliser dans la modélisation. Que ce soit de la discrétisation de variables quantitatives, la normalisation, la standardisation ou la création de nouvelles variables. Il y a souvent une grande quantité de travail, et cette partie est souvent reprise et modifiée en fonction des résultats obtenus lors de la phase d'après, la modélisation. La modélisation consiste à appliquer des techniques statistiques aux données. Le choix du modèle dépend de l'objectif et du type de données utilisées. L'étape implique de nombreuses itérations, ou parfois la phase de préparation de données est reprise également. Ensuite, une fois que le modèle développé est suffisamment performant, vient l'étape d'évaluation. Il s'agit de vérifier que les objectifs définis plus tôt ont été atteints. C'est ici qu'une conclusion est développée. La phase finale consiste à déployer le modèle et à finaliser le projet au moyen d'un rapport complet rendu au client.

Ce projet comporte cependant quelques nuances. Tout d'abord, une étape supplémentaire est nécessaire. Les données ne sont pas accessibles immédiatement et nécessitent une étape de récupération des données. Elles sont accessibles en ligne et doivent être récupérées à l'aide d'un outil de scraping. Ensuite, toute une dimension d'analyse et de visualisation est intégrée au projet. Cette partie est présente à la fois dans les premières phases, pour aider à la compréhension des données, mais aussi dans la phase finale pour présenter des résultats. Pour les phases d'évaluation et de déploiement, il y a également de petites différences. L'évaluation passe aussi par la proposition de visuels dans un dashboard Power BI pour conclure ce projet, cette partie étant l'un des objectifs du projet. Pour le déploiement, il s'agit ici uniquement de remettre le projet pour l'évaluation finale, ce qui ne nécessite pas de mise en place et de communication particulière.

V. Compréhension métiers

La natation et le monde sportif en général sont des domaines assez complexes où chaque détail compte dans l'objectif d'améliorer les performances. Que ce soit dans le très haut niveau, où les gains marginaux font la différence, ou à d'autres niveaux moins exigeants, car la motivation d'avoir des objectifs réalisables y est précieuse. Beaucoup d'investissements sont faits pour améliorer les équipements et la gestion des entraînements, mais il manque encore d'outils permettant de visualiser et anticiper les performances des nageurs. Pour pallier cela, une piste intéressante est envisageable dans l'analyse des résultats, accessibles en grande quantité en ligne sur le site de [swimrankings](#). Analyser ces résultats pourrait apporter une vision globale sur l'évolution des performances de natation et améliorer le suivi des nageurs.

Les objectifs définis sont les suivants :

1. Créer et modéliser une base de données pour enregistrer les données
 - a. Des nageurs : nom, prénom, club, sexe, année de naissance, nationalité
 - b. Des résultats : nageur, distance, style, piscine (50m ou 25m), temps, points FINA, date, lieu
2. Construire un flux ETL pour enregistrer les propres données dans la base de données
 - a. Retirer les colonnes inutiles
 - b. Transformer les types de données
 - c. Effacer les données incomplètes
3. Proposer un dashboard à l'aide d'un outil BI pour présenter les performances des membres d'un club
 - a. Voir les performances collectives pour identifier des améliorations possibles dans la planification des entraînements
 - b. Voir les performances individuelles des membres pour suivre leur évolution et voir des objectifs réalisables à court terme
4. Proposer un dashboard à l'aide d'un outil BI pour présenter et comparer les performances de nageurs
 - a. Pour comparer son niveau aux résultats d'autres nageurs belges à chaque âge
 - b. Pour identifier des objectifs à court ou moyen terme réalisables
5. Créer un modèle de prédiction de performances d'un nageur sur une course afin d'identifier des objectifs atteignables
 - a. Choisir la technique d'apprentissage
 - b. Appliquer la technique aux données existantes
 - c. Réaliser des prédictions sur des nouvelles données

Les autres parties de la mise en place d'une structure organisationnelle du projet ne sont pas réellement d'application pour ce travail de fin d'études. Aucune autre personne n'est impliquée, que ce soit dans l'organisation ou dans la réalisation. Cela a des avantages, comme la réduction des dépendances et du temps nécessaire à organiser la communication. À l'inverse, cela peut rendre l'analyse incomplète et rendre la compréhension plus difficile si les connaissances dans le domaine sont insuffisantes.

VI. Récupération des données

Cette étape est une étape intermédiaire qui, en général, n'a pas lieu d'être dans un projet data. La plupart du temps, les données sont déjà enregistrées au sein de l'entreprise du client, et seul l'accès à celles-ci est nécessaire. Dans certains cas, comme ici, les données ne sont pas à disposition immédiate dans une base de données ou un datawarehouse. Elles n'appartiennent pas à une entreprise spécifique, mais sont accessibles publiquement sur le site internet www.swimrakings.net. Il faut donc passer par un outil de Web Scraping.

Belgium Messieurs, Cat. générale			100m Libre						Grand bassin (50m) Meilleurs à vie		
Dernière mise à jour 11 mai 2025 - 0:01											
Afficher les meilleurs temps Premier <<250 <<50 <<25 Depuis le rang: 1			GO 25>> 50>> 250>> Rangs de 1 à 9035								
Nom, Prénom	A.n.	nat.	Club	Temps	Pts.	Rang dans class.	Date	Ville (Pays)	BEL	Région	Club
TIMMERS, Pieter	1988	BEL	Zwemclub Brabo Antwerpen	47.80	942	1. 1. 1.	10 aoû 2016	Rio (BRA)			
SURGELOOSE, Glenn	1989	BEL	Zwemclub Brabo Antwerpen	48.64	894	2. 2. 2.	19 mai 2016	London (GBR)			
DEKONINCK, Dieter	1991	BEL	Zwemclub Brabo Antwerpen	48.79	885	3. 3. 3.	29 jul 2012	London (GBR)			
GRANDJEAN, Yoris	1989	BEL	Liege Natation	48.82	884	4. 1. 1.	12 aoû 2008	Beijing (CHN)			
HENVEAUX, Lucas	2000	BEL	Liege Natation	48.96	876	5. 2. 2.	NEW 25 avr 2025	Antwerpen			
AERENTS, Jasper	1992	BEL	Koninklijke Brugse Zwem- & Reddingskring	49.08	870	6. 1. 1.	28 fév 2016	Antwerpen			

Figure 1 : Swimstats - Résultats Hommes Belge

La récupération des données est une tâche compliquée qui nécessite une compréhension des langages web (HTML et CSS principalement) et une solution pour récupérer ces données. Pour cela, il est possible de passer par des outils, parfois payants, déjà existants. L'avantage de ces applications est qu'il suffit de les paramétriser légèrement pour automatiser la récupération des données, et tout cela sans avoir besoin de comprendre réellement la logique du développement web. Ils ne permettent cependant pas une aussi grande liberté qu'en utilisant des librairies Python, qui servent de base pour décomposer et identifier les zones de données dans une page internet. L'adaptabilité et la souplesse de Python en font une solution bien plus appropriée pour ce travail. L'automatisation à l'aide de librairies comme 'Requests' et 'BeautifulSoup' permet de récupérer les données sur un grand nombre de pages web et offre une meilleure gestion des erreurs.

Toutes ces données doivent ensuite être enregistrées dans une base de données afin de permettre leur exploitation structurée. Pour cela, une phase de modélisation préalable est nécessaire afin de concevoir un schéma logique et normalisé. L'outil Draw.io a été utilisé pour créer le diagramme du modèle relationnel. Sur base de ce schéma, plusieurs technologies peuvent être utilisées dans la mise en place de la base de données. Microsoft SQL Server est une solution intéressante et cohérente, notamment en raison de sa complémentarité avec SQL Server Integration Services

(SSIS), un outil d'ETL également développé par Microsoft. Ce dernier sera utilisé pour automatiser la transformation et le chargement des données brutes vers la base de données modélisée et normalisée. C'est finalement PostgreSQL qui a été utilisé dans le cadre de ce projet. Cette technologie open source présente une prise en main plus simple et une plus grande flexibilité. Elle est également bien adaptée à un déploiement sur serveur, ce qui facilite son intégration dans un environnement distant, nécessaire pour le projet. En outre, l'exportation des données au format CSV y est particulièrement aisée, ce qui est un atout supplémentaire dans l'optique de vouloir récupérer ces données en local facilement.

Dans le but de pouvoir avancer sur d'autres parties du projet pendant l'exécution des scripts de récupération de données, le scraping a été réalisé sur un serveur distant. Cela a permis, non seulement de libérer du temps et des ressources sur un poste de travail local, mais aussi d'assurer une meilleure stabilité de connexion. Il existe ici aussi un grand nombre de services d'hébergement. Afin de réduire les coûts à zéro, DigitalOcean est la solution idéale pour les étudiants. Ils proposent une formule gratuite largement suffisante pour réaliser ce type de projets. Pour faciliter encore davantage la gestion, le projet est passé par la création de conteneurs Docker. Ils permettent de faire tourner des scripts dans des environnements indépendants sur un même serveur, et de faciliter la gestion des erreurs.

a. Web scraping

Dans un premier temps, l'objectif est d'enregistrer les informations relatives aux nageurs et nageuses belges. La course la plus pertinente pour cela est le 100m nage libre. Étant la course la plus nagée et le style de nage le plus commun, ce sont ces résultats qui expliquent le mieux l'évolution d'un ou d'une nageur.se. Ces informations sont affichées sur 2 URLs distinctes : L'une pour les hommes et l'autre pour les femmes. Chaque page ne contient que le meilleur temps de 25 sportifs, dont la première place est identifiée dans le lien de recherche. Il faut donc créer un programme itératif, qui incrémente la valeur de départ dans l'URL afin de récupérer les données page par page de tous les nageurs et nageuses belges. Une première réflexion sur la pertinence des données est nécessaire. Il y a dans le script une fonction « sleep », qui ajoute un temps d'attente, permettant d'éviter que le site bloque les requêtes réalisées depuis une même source trop rapidement. C'est une protection standard pour éviter des attaques de type DDOS, qui peut faire dysfonctionner le site. Le problème est que cela augmente considérablement la durée d'exécution et pousse à réfléchir à une limitation du nombre de personnes pertinentes pour l'analyse. Sur base des résultats disponibles sur Swimrankings, il y a plus de 9000 hommes et 9000 femmes ayant réalisé au moins un 100m nage libre en compétition. Les performances les plus lentes parmi elles n'ont pas de valeur ajoutée pour la suite du projet. Ces résultats sont soit réalisés par des jeunes, ce qui ne permet donc pas de suivre une évolution sur plusieurs années, soit par des sportifs moins réguliers, qui ne sont pas représentatifs d'une évolution standard. La décision a donc été prise de limiter les résultats sur base des points FINA (cf. 0 compréhension des données). Seuls les

nageurs ayant un résultat atteignant au moins 400 points FINA sont gardés, ce qui réduit le nombre total de nageurs à un peu plus de 6500.

Avant de passer à une nouvelle partie de script, pour récupérer l'entièreté des résultats, il faut encore choisir quels résultats sont pertinents à enregistrer. Il y a en Belgique un total de 17 disciplines différentes, réparties dans 4 (5 en comptant le 4 nages) styles (nage libre (crawl), papillon, dos et brasse) et des distances allant de 50m à 1500m. Parmi celles-ci, quelques-unes sont plus souvent courues. Principalement les 100m des 4 nages principales, 200m et 400m nage libre et le 200m

Meilleurs temps:	Sélectionner...	Classements personnels:	100m Libre	Tous les résultats:	Sélectionner...
Personal rankings for 100m Libre					
Grand bassin (50m)					X
53.62	896	9 fév 2025	Antwerp		
54.18	869	16 mar 2025	Edinburgh (GBR)		
54.51	853	10 fév 2023	Antwerpen		
54.52	853	15 avr 2025	Stockholm (SWE)		
54.56	851	23 avr 2023	Antwerpen		
54.72	843	10 fév 2023	Antwerpen		
54.78	841	23 avr 2023	Antwerpen		
54.82	839	24 jan 2025	Genève (SUI)		
54.99	831	16 mar 2025	Edinburgh (GBR)		
55.06	828	9 fév 2025	Antwerp		
55.19	822	3 déc 2022	Rotterdam (NED)		
55.24	820	3 déc 2022	Rotterdam (NED)		
55.34	815	9 jul 2022	Otopeni (ROU)		
55.36	814	24 jan 2025	Genève (SUI)		
55.42	812	15 avr 2025	Stockholm (SWE)		
55.48	809	8 jul 2022	Otopeni (ROU)		
55.51	808	5 jan 2025	Wezenberg		
55.68	800	11 aoû 2022	Rome (ITA)		
55.82	794	8 jul 2022	Otopeni (ROU)		
55.89	791	27 jul 2023	Fukuoka (JPN)		
55.90	791	29 mai 2024	Barcelona (ESP)		
55.95	789	11 aoû 2022	Rome (ITA)		
56.00	787	5 déc 2021	Castellon (ESP)		
56.01	786	5 déc 2021	Castellon (ESP)		
56.01	786	24 avr 2022	Antwerpen		
Petit bassin (25m)					X
52.61	871	15 déc 2024	Copenhagen (DEN)		
52.78	862	10 nov 2024	Gent		
53.12	846	10 nov 2024	Gent		
53.15	845	27 oct 2024	Aachen (GER)		
53.50	828	27 oct 2024	Aachen (GER)		
53.78	815	15 déc 2024	Copenhagen (DEN)		
54.71	774	16 oct 2022	Aachen (GER)		
54.84	769	16 oct 2022	Aachen (GER)		
54.89	767	18 déc 2022	De Sprint		
54.95	764	8 mai 2022	Hasselt		
55.01	762	20 oct 2024	Drachten (NED)		
55.53	741	10 nov 2019	Sint Amandsberg		
55.90	726	10 nov 2019	Sint Amandsberg		
57.55	665	18 nov 2018	Nijlen		
58.82	629	18 nov 2018	Nijlen		
59.13	613	11 mar 2018	Bree		
59.48	602	19 nov 2017	Nijlen		
1:00.08	585	19 nov 2017	Nijlen		
1:00.20	581	15 oct 2017	Overpeelt		
1:02.26	525	4 jun 2017	Bree		
1:04.50	472	18 déc 2016	Bree		
1:07.71	408	5 jun 2016	Bree		
1:09.09	384	6 mar 2016	Maasmechelen		
1:10.24	366	27 déc 2015	Mol		
1:10.89	356	21 fév 2016	Bree		

Figure 2 : Swimstats - Résultats Individuels 100m nage libre

4 nages. Chaque nageur a une page reprenant tous les résultats de chaque course séparément, dans un tableau de 2 colonnes. Une pour les résultats en grand bain (piscine de 50m) et une pour les résultats en petit bain (25m). Cette nuance est importante à prendre en compte aussi lors de la récupération de ces données de résultats. Il faut donc créer un script qui itère sur les 7 différentes courses sélectionnées pour chaque nageur et qui récupère les résultats de toutes les compétitions en gardant bien la nuance entre le bassin de 25m et de 50m. Une information supplémentaire qui est parfois disponible est le ou les temps intermédiaire(s) de la course. Cette information se retrouve dans un autre tableau qui ne s'affiche que lorsqu'un nageur survole un résultat. Comme cette information est facilement accessible, le script fera en sorte d'enregistrer ces informations également.

b. Base de données

Avoir des données est une chose, s'assurer de l'intégrité, de la normalisation et de la consistance de celles-ci en est une autre. Enregistrer ces données dans une base de données relationnelle permet de solutionner ces problèmes potentiels. Pour cela, il faut en premier lieu un schéma cohérent, respectant les principes de la normalisation, ce qui réduit les risques de redondance et d'incohérences.

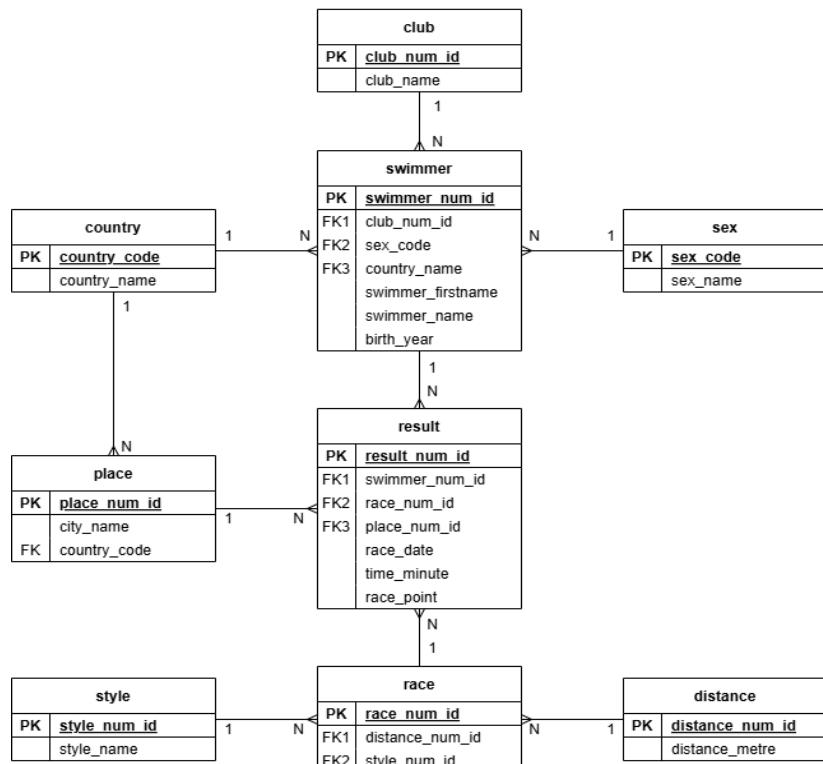


Figure 3 : Schéma relationnel

Le schéma se rapproche d'un modèle en flocon. La table centrale « result » regroupe les faits, les résultats de compétition, observés. Elle est liée à plusieurs tables de dimension, dont la principale est la table « swimmer ». Cette dernière référence également d'autres tables de dimension qui sont principalement des tables de code, qui répondent aux problèmes de redondance. La table « race » est une table dite d'intersection. Elle est nécessaire pour normaliser et éviter les doublons dans le cas d'une relation « Many to Many ». Cela permet aussi une meilleure flexibilité dans le cas où de nouvelles épreuves devaient être ajoutées.

Ce schéma est ensuite implémenté dans PostgreSQL. Pour cela, le script de création de tables et de clés primaires et étrangères est développé de manière à respecter les contraintes du diagramme, ainsi que les contraintes liées aux données. Chaque colonne est liée à un domaine qui limite les valeurs dans un certain type de données afin de s'assurer qu'il n'y ait pas de valeurs incorrectes enregistrées.

c. Serveur

La dernière étape de la mise en place de la récupération automatisée des données consiste à encapsuler les scripts dans des images Docker. Un fichier Dockerfile permet de définir les ressources nécessaires, les bibliothèques à installer, les variables d'environnement et le comportement du conteneur lors de l'exécution. Pour faciliter la communication entre les conteneurs avec le script python et celui de la base de données PostgreSQL, un réseau Docker est d'abord créé via la commande : `docker network create swimstats_nw`

Les deux images déployées sur le serveur sont d'une part pour la base de données (swimstats:db) et d'autre part pour le script de scraping python (swimstats:py). Une fois créées à l'aide de la commande « `docker build -t <nom_image>` » et ensuite mises à disposition sur Docker même avec « `docker push <nom_image>` ». Elles sont dès lors récupérables depuis le serveur à l'aide de la commande « `docker pull <nom_image>` », et exécutables en utilisant cette commande : « `docker run -network swimstats_nw -name <nom_conteneur> -d -t <nom_image>` ». Les paramètres de cette dernière commande permettent respectivement d'identifier le réseau dans lequel il doit être exécuté, le nom associé au conteneur, le lancement en mode 'daemon' (pour que le conteneur s'exécute en tâche de fond), et dernièrement le nom de l'image à exécuter. « `docker logs <nom_conteneur>` » est la commande à lancer pour vérifier le statut du script et voir si des logs peuvent expliquer une potentielle erreur. Dernièrement, 2 commandes similaires ont été utilisées pour extraire les données brutes du conteneur PostgreSQL vers des fichiers CSV :

- `docker exec -it -u <user> <nom_conteneur> psql -d <nom_database> -c "COPY (SELECT * FROM races) TO STDOUT WITH CSV HEADER" > races.csv`
- `docker exec -it -u <user> <nom_conteneur> psql -d <nom_database> -c "COPY (SELECT * FROM swimmers) TO STDOUT WITH CSV HEADER" > swimmers.csv`

Une subtilité importante dans cette configuration est l'utilisation de 2 serveurs distincts. Chaque serveur est dédié à la récupération des résultats pour un sexe. Cette approche permet de paralléliser l'exécution des scripts, ce qui réduit considérablement le temps nécessaire pour récupérer toutes les données. De plus, cette séparation sur 2 serveurs avec des adresses IP différentes permet de répartir la charge des requêtes et de limiter les risques de blocage de la part du site.

root@ubuntu-s-1vcpu-2gb-ams3-01:~# docker ps -a						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
485085a50d0c	swithan/swimstats:py	"python s.py"	3 seconds ago	Up 1 second		py
ef6f71971230	swithan/swimstats:db	"docker-entrypoint.s..."	5 months ago	Up 5 months	5432/tcp	db

Figure 4 : Liste des conteneurs actifs sur un serveur

Pour récupérer les données exportées dans les fichiers CSV, une dernière commande doit être exécutée sur l'ordinateur local, en utilisant `scp`, un protocole de transfert de fichiers.

```
scp <user>@<server_ip>:<path_to_csv> <path_to_output>
```

VII. Compréhension des données

Explorer et analyser les données permet de dégager des observations générales sur les données. Tout d'abord, il est important de comprendre chaque colonne des fichiers de données récupérés lors de la phase précédente. La plupart des informations sont assez explicites, comme le nom, le prénom, la date de naissance, le sexe, la nationalité et le dernier club d'un nageur, associé à son ID spécifique. Pour les données relatives aux résultats, il y a davantage de subtilités. Premièrement, il y a la colonne avec l'identifiant du nageur, permettant de relier les 2 tables entre elles. Ensuite, toutes les informations relatives au résultat : la distance et le style, la date, le lieu (ville et pays), la longueur de la piscine, le temps réalisé (en secondes) et le nombre de points FINA auquel le résultat correspond. Cette dernière est un nombre de points (P) calculé sur la base du temps réalisé (T) et d'un temps de référence (B) à l'aide de la formule suivante : $P = 1000 * (B / T)^3$. Les temps de référence sont redéfinis chaque année sur la base des derniers records du monde ratifiés. Il y a des temps de référence pour chaque course, différents pour les hommes et les femmes, et également séparés pour les courses en grand bassin (piscine de 50m) et en petit bassin (piscine de 25m). La phase de récupération des données a permis de récupérer un total de 6651 nageurs qui, ensemble, ont réalisé 1480181 résultats. Comment ces résultats sont-ils distribués selon l'âge ou le sexe du nageur ? Quel est le plus grand club du pays en termes de nombre de nageurs ? Est-ce que ce club est aussi celui avec le plus grand nombre de courses par nageur ? Au niveau des courses, quelles sont les disciplines et distances les plus populaires ? Comment évoluent les résultats en fonction de l'âge du nageur ?

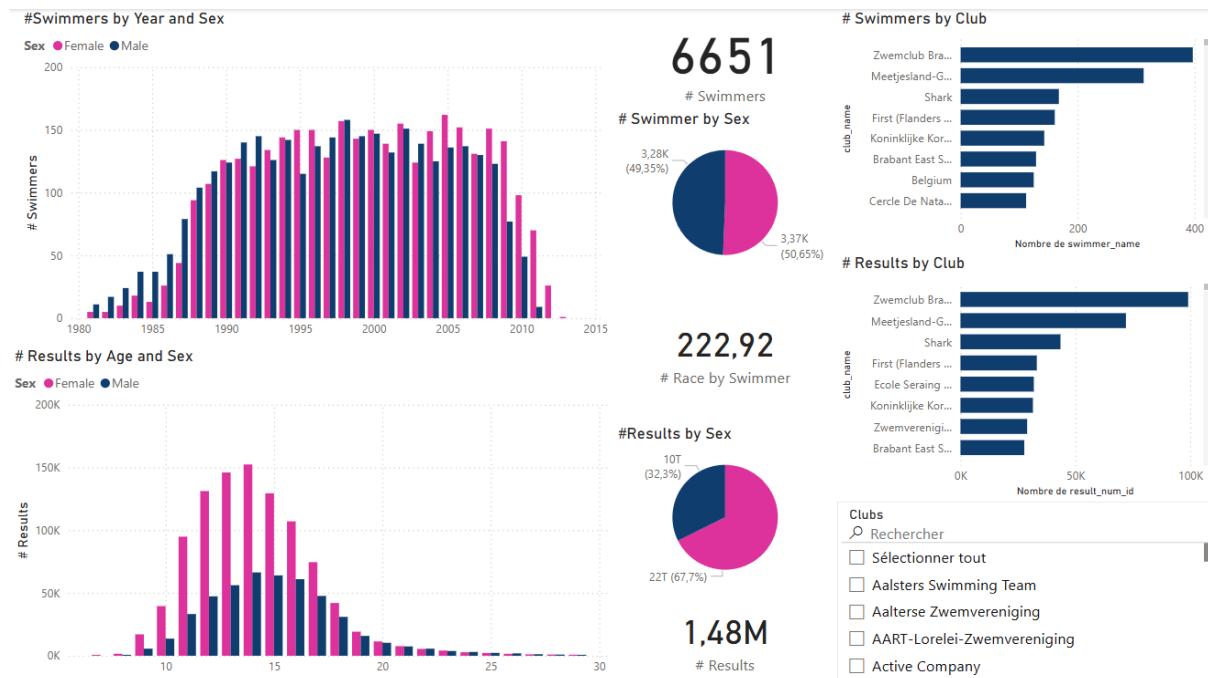


Figure 5 : Dashboard nageurs & clubs

a. Visualisation

Un premier dashboard réalisé sur Power BI permet de distinguer la distribution du nombre de nageurs par année de naissance et par sexe, le nombre de nageurs par club et le nombre de courses réalisées par âge et sexe. Des KPI reprennent le nombre de résultats et de nageurs, ainsi qu'une mesure calculée permettant d'identifier le nombre de courses par nageur. On peut premièrement constater que, malgré un nombre de nageurs assez similaire dans les 2 sexes, le nombre de courses réalisées par les femmes est 2 fois plus important que pour les hommes. Cette information est visible dans les pie charts, mais aussi déductible dans les bar charts. La distribution de ces mêmes KPI par club suit une distribution globalement similaire pour le nombre de nageurs et de résultats, expliquée par le troisième KPI indiquant le nombre de courses par nageur.

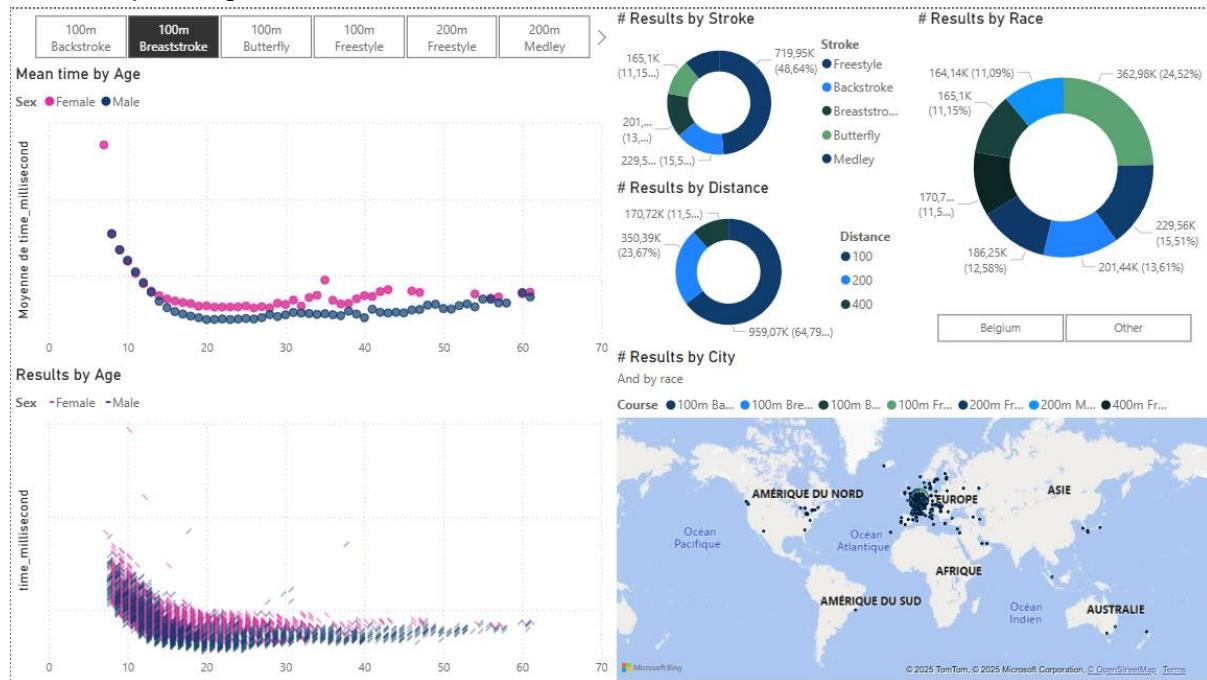


Figure 6 : Dashboard résultats

Un second dashboard présente d'une part la distribution et l'évolution des résultats avec l'âge pour chaque course séparément. Ces graphes présentent tous une évolution rapide à un jeune âge, avant de se stabiliser et de finir par augmenter légèrement. Des donut-charts permettent de voir la distribution des courses par style et par distance afin d'indiquer la ou les courses les plus communes. Dans ce dashboard, une carte présente finalement les lieux où les compétitions ont eu lieu, avec un filtre permettant de se concentrer sur les compétitions ayant eu lieu en Belgique. On aperçoit ici quelques manquements liés à la qualité des données. Certains lieux comme Anvers indiquent plusieurs points au même endroit, sans assembler les valeurs. Cela est dû à un manque de normalisation des valeurs. Les compétitions à Louvain ne sont pas associées avec celles réalisées à Leuven.

b. Statistiques descriptives

Afin d'améliorer encore la compréhension des données, une analyse statistique descriptive est réalisée. Les statistiques calculées varient en fonction du type de données. Pour les données quantitatives, un graphe de distribution des temps est réalisé. Cela permet d'identifier des valeurs aberrantes, de voir la moyenne, les minimums et maximums ainsi que les écarts interquartiles. Une information surprenante visible ici est que, bien que le meilleur temps de chaque course soit toujours réalisé en piscine de 25m, la distribution moyenne des courses réalisées dans des longueurs de 50m est plus rapide et l'écart interquartile est plus petit.

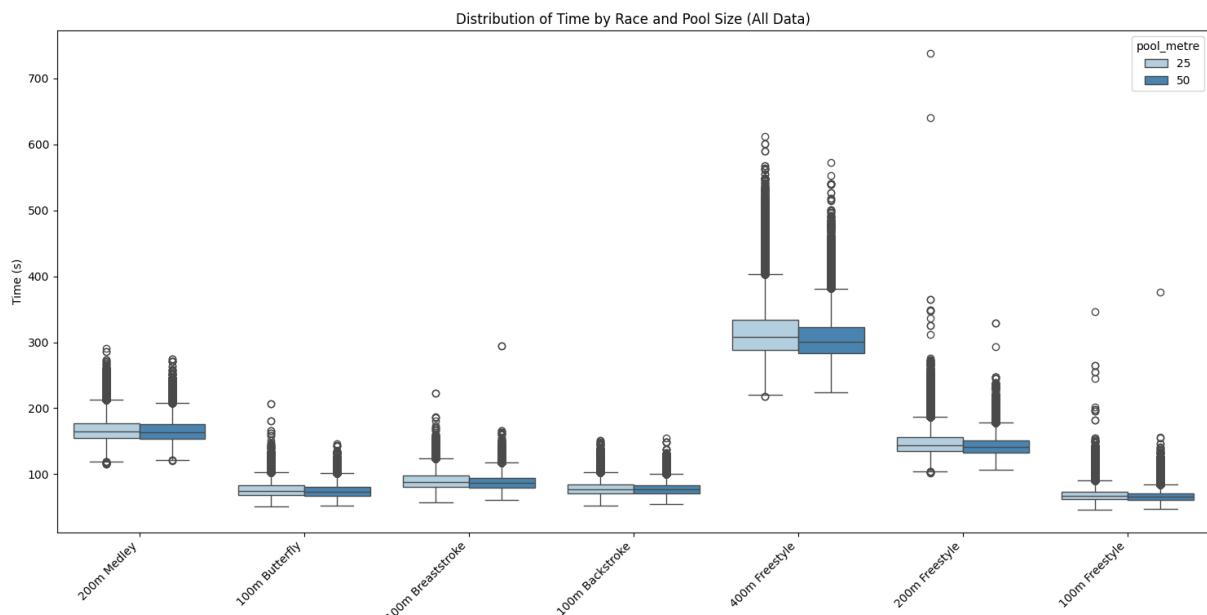


Figure 7 : Distribution des temps par course

Pour les données temporelles comme la date de la course ou l'année de naissance, les statistiques suivantes sont identifiées :

Colonne	# valeurs	Min	Max	Mode
result_date	3801	1988-09-23	2025-03-30	2005-07-28 (3678)
birth_year	51	1956	2013	1998

On peut dès lors affirmer que les données traitent de résultats réalisés entre 1988 et 2025, pour des nageurs nés entre 1956 et 2013. Les données qualitatives comme le style de nage ou le nom du club ne nécessitent pas de statistiques approfondies. Elles sont déjà expliquées à l'aide des dashboards Power BI présentés plus tôt.

VIII. Préparation des données

La qualité des données est une condition indispensable pour pouvoir créer un modèle prédictif performant. Avant cela, il est nécessaire de passer par une phase de préparation des données. Lors de cette étape, la première tâche importante est de nettoyer les données non normalisées, de gérer les données manquantes et de supprimer les données redondantes. Cela est réalisé à l'aide d'un outil d'ETL. SSIS, l'outil d'ETL développé par Microsoft, en plus d'être étudié dans le cadre du bachelier de spécialisation, propose une interface visuelle très intuitive. Chaque étape est clairement identifiable, ce qui facilite la gestion d'erreurs et l'adaptabilité. Sur base de ces données propres et exploitables, la seconde partie implique la création de nouvelles variables. Ces nouvelles informations permettent d'augmenter la précision des prédictions d'un modèle. Pour effectuer cela, Python et sa librairie Pandas, sont la solution privilégiée. La principale raison est que la modélisation passe également par l'utilisation de ce langage de programmation et que revenir à la phase de « feature engineering » (création de variables) est très souvent nécessaire. Cela facilite donc l'itération de ces 2 étapes.

a. ETL

Une approche structurée et réfléchie est nécessaire pour mettre en place un bon flux ETL. Sur base des données brutes enregistrées dans divers fichiers CSV et du schéma de base de données, chaque étape de transformation peut être mise en place. En premier lieu, toutes les données en CSV sont transférées vers des tables de base de données temporaires. La seule modification faite lors de cette étape est la séparation des informations du lieu de la compétition. Les informations de localisation varient en fonction du pays dans lequel la course a eu lieu. Dans le cas où la compétition s'est déroulée en Belgique, seul le nom de la ville est partagé. Pour les courses parcourues en dehors de la Belgique, il y a généralement un code de pays noté dans cette même colonne. Une séparation est donc faite entre ces deux informations, pour créer une colonne ville et une colonne pays. Pour compléter les données, une valeur n'ayant pas de pays spécifié originellement se voit attribuer le pays belge automatiquement.

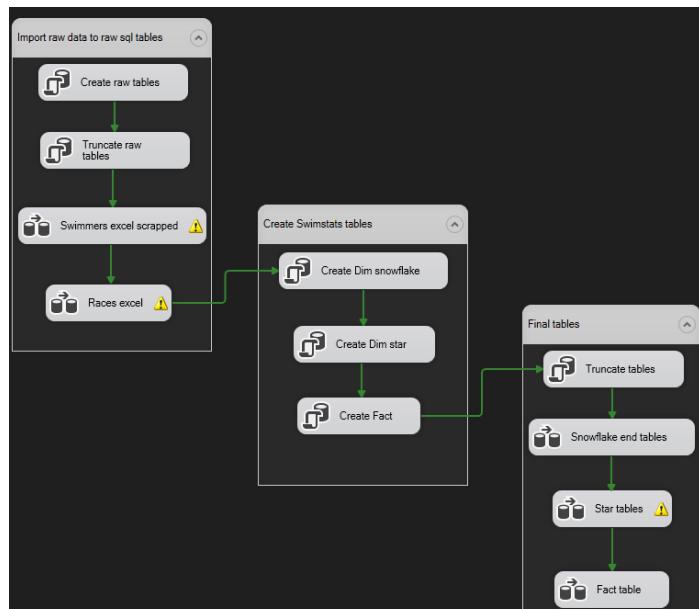


Figure 8 : Flux de données global

Le second groupe rassemble toutes les tâches d'exécution des scripts de création des tables modélisées et normalisées. Comme illustré dans le chapitre Base de données, un modèle en flocon a été préféré dans le cadre du projet. Ceci implique 3 niveaux de tables. La table centrale reprend les faits, dans ce cas les résultats. Directement connectées à la table de fait, sont les tables de dimension principales. Celles-ci réfèrent généralement encore à des tables de dimension secondaires, ou périphériques. C'est par ces dernières que l'exécution des scripts de création de tables démarre car elles ne comportent aucune clé étrangère, donc aucune dépendance à d'autres tables. Les tables de dimension principales viennent ensuite en référençant les tables déjà créées. Pour finir, le script de création de la table des résultats (table de faits) est exécuté, avec la création des clés étrangères associées.

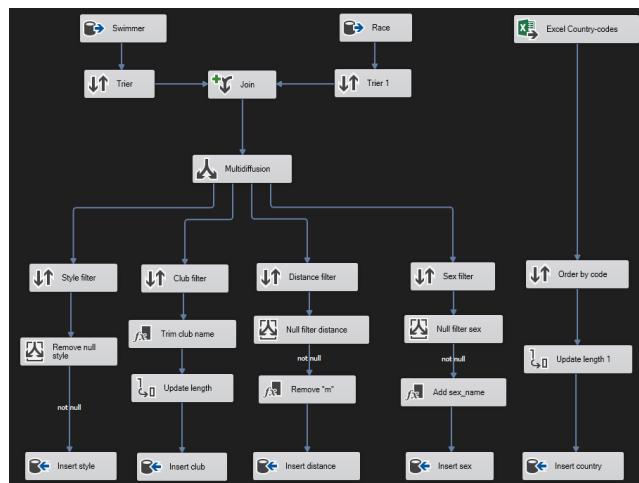


Figure 9 : Flux des tables de dimensions périphériques

Insérer les données dans les nouvelles tables créées demande davantage de travail. En reprenant l'ordre d'exécution des scripts de création de tables, chaque table peut être remplie sur base des données brutes enregistrées lors de la première étape du flux ETL. Afin de respecter les contraintes des tables normalisées qui permettent d'assurer une qualité des données, plusieurs transformations sont nécessaires. Les tables de dimension périphériques sont toutes des tables de code. Elles reprennent toutes les valeurs uniques possibles pour les styles et les distances de course, les noms des clubs et le sexe des nageurs extraits depuis les tables de données brutes. La table « country » est un peu particulière. Celle-ci ne récupère pas les données extraites du site web, mais depuis un fichier Excel supplémentaire. Ce fichier comporte une liste des pays avec leur code correspondant (ex. BEL pour Belgique). Le flux illustré à la Figure 9 : Flux des tables de dimensions périphériques montre le processus d'extraction, de filtrage et de transformation appliqué à chaque concept. Les données brutes sont récupérées dans la base de données et assemblées. Pour chaque dimension, un filtre récupère uniquement les valeurs distinctes pour chacune des colonnes pertinentes. Des transformations spécifiques sont appliquées afin de gérer correctement le type de données (Ex. : les distances sont enregistrées avec les unités dans la structure de données non transformée, information qui doit être retirée). Chaque flux se termine par une tâche de destination pour remplir la table correspondante.

Pour les tables de dimension principales et la table de fait, le flux de données SSIS nécessite une approche particulière. Les lignes de données sont récupérées à l'aide de requêtes SQL spécifiques. Ces requêtes permettent de sélectionner uniquement les colonnes utiles, de joindre les tables sur base des clés étrangères facilement et d'appliquer des filtres précis. C'est le cas du code SQL ci-dessous, qui permet de récupérer toutes les courses distinctes présentes dans les données brutes et relie les informations de distance et de style de course aux tables de dimension secondaires. La commande SQL précise que les seules colonnes finalement récupérées sont les identifiants de style et de distance, qui sont les informations à récupérer et insérer dans la table « race ».

```

SELECT DISTINCT
    d.distance_num_id,
    s.style_num_id
FROM public.race AS r
JOIN swimstats.distance AS d
    ON CAST(REPLACE(r.distance, 'm', '') AS INTEGER) =
d.distance_metre
JOIN swimstats.style AS s
    ON r.stroke = s.style_name;

```

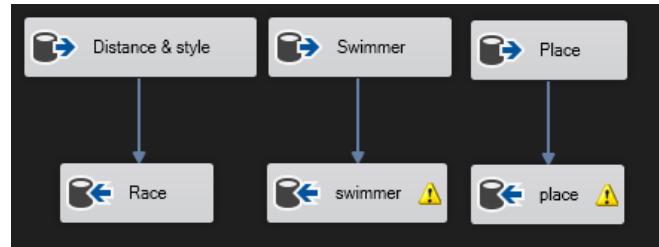


Figure 10 : Flux des tables de dimensions principales

b. Feature engineering

Dans le but d'optimiser les performances du modèle, créer des « features » ou colonnes supplémentaires est indispensable. Ces nouvelles informations permettent de mieux représenter les tendances et les comportements identifiables dans les données d'origine. Un modèle de réseau de neurones fonctionne beaucoup mieux lorsque beaucoup de variables non corrélées sont ajoutées. Des premières variables calculées directement sont facilement identifiées. L'âge auquel le résultat est réalisé est dépendant de l'année de naissance et de la date de la course. Avec une réflexion plus approfondie, en natation, l'âge est basé sur une saison sportive allant de septembre à aout, ce qui implique une modification du calcul de l'âge : l'année de la course est calculée sur la saison. Une course en mars 2025 est donc réalisée dans la saison 2024, tout comme une course en octobre de cette dernière année. Pour d'autres colonnes, une normalisation des informations est nécessaire. Par exemple, le sexe (M ou F) ou la longueur de la piscine (25m ou 50m) sont transformés en unités binaires (0 ou 1). Les données catégoriques sont adaptées pour qu'un modèle puisse en extraire de l'information pertinente. En cherchant à optimiser encore les performances du modèle, les résultats précédents, sur la même course, sont également ajoutés dans une nouvelle colonne, tout comme le temps écoulé depuis la dernière course effectuée. L'information de l'âge lors de la première course peut également être un facteur pertinent pour l'entraînement du modèle. La dernière

colonne ajoutée est une colonne avec un calcul complexe. À l'aide d'une régression linéaire, la tendance d'évolution sur les 3 dernières courses est calculée. C'est une information cruciale qui permet d'expliquer l'évolution du temps d'un nageur sur un très court terme. Si cette valeur est négative, cela indique une progression. À l'inverse, un coefficient de pente positif indique un manque de forme récent.

```
def compute_trend(group, window=3):
    trends = []
    for i in range(len(group)):
        if i < window - 1:
            trends.append(np.nan)
        else:
            y = group['time_millisecond'].iloc[i - window
+ 1:i + 1].values.reshape(-1, 1)
            x = np.arange(window).reshape(-1, 1)
            model = LinearRegression().fit(x, y)
            slope = model.coef_[0][0]
            trends.append(slope)
    return pd.Series(trends, index=group.index)
```

IX. Modélisation des données

Avant de passer à la mise en place de techniques statistiques sur les données, il est important de s'assurer de comprendre le cadre dans lequel il s'inscrit. L'objectif du projet est de prédire les résultats de compétition, indiquant qu'une technique de prédiction d'estimation doit être appliquée. Cette technique s'inscrit dans le cadre d'un apprentissage supervisé, avec des prédicteurs (colonnes X) et une valeur correspondante Y, le résultat de la course. De plus, il y a une relation claire entre les variables explicatives et les résultats. L'âge en est le principal exemple. Dans ce contexte, les modèles de régression linéaire ou polynomiale et les réseaux de neurones sont les techniques les plus adaptées.

Pour confirmer ces hypothèses, à l'aide des visualisations réalisées en amont, il est simple d'identifier une tendance d'évolution des résultats de natation en fonction de l'âge. Avec une évolution rapide au début, et une tendance atteignant une limite minimale avant d'augmenter légèrement à nouveau. Ceci indique qu'une régression polynomiale pourrait expliquer la relation et généraliser l'évolution du temps d'un sportif. Pour s'assurer de trouver le meilleur modèle, configurer un réseau de neurones et l'optimiser est la seconde technique appliquée aux données.

Comme expliqué lors de la phase de préparation des données et du « Feature engineering », la création d'un modèle performant passe par beaucoup d'adaptations et de changements au niveau de la création de variables. Il y a ensuite de nombreuses itérations nécessaires pour trouver la configuration du modèle la plus optimale.

a. Régression polynomiale

L'implémentation d'un modèle de régression polynomiale se fait en plusieurs étapes. Sur base des données importées dans le script python et des nouvelles colonnes créées lors de la phase de feature engineering, une sélection de variable est réalisée. Les colonnes avec les variables prédictives sont assemblées, et la colonne cible est gardée séparément. Ensuite, les données sont séparées en groupe de données d'entraînement, qui sera utilisé pour entraîner le modèle, et le groupe de test, qui sert de vérification des résultats de la phase d'entraînement. L'entraînement d'un modèle de régression est un processus itératif. Une valeur de pente (w) et d'ordonnée à l'origine (b) est attribuée aléatoirement. À l'aide de l'algorithme du gradient descent, des itérations sont réalisées sur les données pour faire varier les valeurs w et b , jusqu'à trouver un optimum local. Cet optimum local est l'endroit où l'erreur d'estimation est la moins importante en moyenne. Sur base des valeurs optimales trouvées lors de l'entraînement avec les données d'entraînement, une vérification est réalisée sur les données de test. Lors de cette étape, il est possible d'identifier des problèmes dans le modèle. Si les résultats de ce modèle ne sont pas suffisants pour les données d'entraînement et les données de test, le modèle pourrait se trouver en situation d'underfitting. Cela signifie que le modèle ne représente pas suffisamment les données et ne peut donc pas être utilisé pour généraliser l'estimation. À l'inverse, si le modèle est très optimisé vis-à-vis des données d'entraînement mais que l'erreur est importante sur les données de test, le risque d'overfitting est grand. Le modèle n'est alors pas généralisable non plus, car il est uniquement applicable aux données sur lesquelles il a été construit.

Pour paramétrier un entraînement de régression linéaire, plusieurs valeurs sont modifiables. Premièrement, les colonnes de variables explicatives. C'est sur ces données que le modèle va se baser pour faire une prédiction de temps. En plus des colonnes de la base de données, il y a l'ajout des colonnes lors de la Préparation des données. Les autres paramètres sont :

1. Les colonnes explicatives pertinentes
2. Les valeurs initiales de w et b (attribution aléatoire)
3. Le coefficient de régularisation (lambda : éviter le surapprentissage)
4. Le nombre d'itérations (iterations : nombre de recalculs de w et b)
5. Le taux d'apprentissage (alpha : taille des sauts pour le gradient descent)

La phase d'apprentissage du modèle de régression polynomiale est très fastidieuse. Le temps requis pour atteindre le nombre d'itérations (ou un arrêt anticipé si le modèle ne s'améliore pas suffisamment entre 2 itérations) est important. Pour quantifier les performances du modèle, un calcul de l'erreur quadratique moyenne (RMSE) permet d'identifier que l'erreur absolue de prédiction d'un temps est de l'ordre de 3,3 secondes. Cela représente une erreur de 5,11% du temps d'une course.

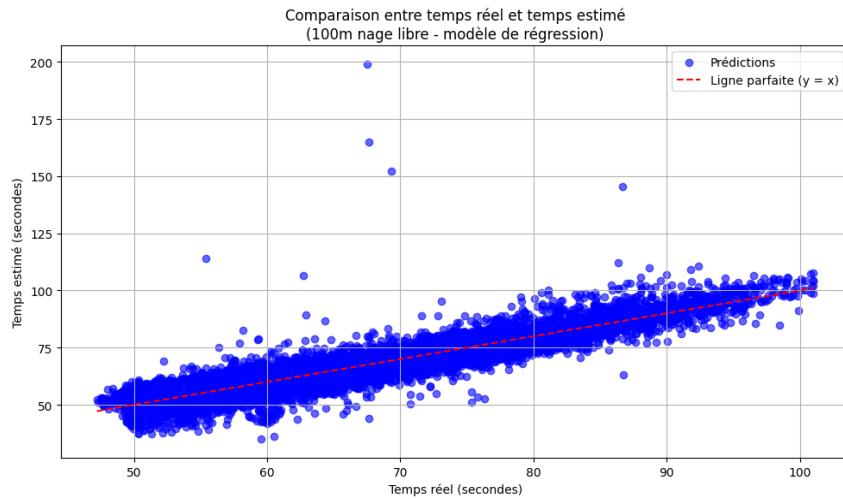


Figure 11 : Résultats modèle de Régression Polynomiale (100m nage libre)

Dans l'idée de comparer ce modèle à un réseau de neurones se basant sur les mêmes informations, les données utilisées pour l'entraînement du modèle se limitent aux résultats obtenus sur le 100m nage libre. Cette course est la plus représentative, tant en termes du nombre total de résultats récupérés qu'en termes de représentation de l'évolution des performances globales d'un nageur. Sur base du meilleur modèle pour cette course, une analyse et un entraînement plus approfondi sont appliqués sur les autres distances et styles de nage.

b. Réseau de neurones

Un modèle de réseau de neurones se compose de plusieurs couches de neurones, reliés à l'aide de fonctions d'activation. La couche d'entrée représente les variables explicatives du modèle. Il y a ensuite une ou plusieurs couches cachées, dont les résultats ne sont pas interprétables par l'humain. Ces couches consécutives utilisant des fonctions d'activation finissent par atteindre la couche de sortie, généralement à l'aide d'une fonction d'activation linéaire dans le cas d'un modèle de prédiction d'estimation. Cette dernière couche est composée d'un seul neurone, qui donne le résultat de la prédiction. L'entraînement du réseau de neurones s'effectue en « epochs », qui représente un cycle de passage des données d'entraînement à travers le réseau de neurones. Lors du cycle, les données suivent en premier lieu une propagation avant : toutes les données passent par les couches du modèle pour donner un résultat. Ensuite, sur base de ce premier passage, une fonction de coût calcule l'erreur, autrement dit l'écart entre le résultat obtenu et le résultat attendu. Vient ensuite un processus itératif de propagation arrière permettant de corriger les poids associés à chaque couche du réseau.

Il y a donc plusieurs paramètres à prendre en compte lors de la mise en place du modèle :

1. Les colonnes explicatives pertinentes
2. Le nombre d'Epochs
3. Le nombre de couches de neurones du modèle
4. Le nombre de neurones par couche du réseau
5. Les fonctions d'activation pour chacune des couches
6. Le batch size (nombre de données par échantillon)

```
model_2 = Sequential([
    Dense(16, activation='relu',
          input_shape=(x_train.shape[1],)),
    Dense(8, activation='relu'),
    Dense(4, activation='relu'),
    Dense(1, activation='linear')
], name='model')
model.fit(
    x_train, y_train,
    epochs=500,
    batch_size=64,
    validation_split=0.1,
    verbose=0
)
```

Ce type de modèle est beaucoup plus paramétrable que le modèle de régression polynomiale créé plus tôt. La structure globale du réseau de neurones (nombre de couches et de nœuds par couche, type de fonction d'activation) est le paramètre le plus manipulé dans l'optique d'optimiser le résultat.

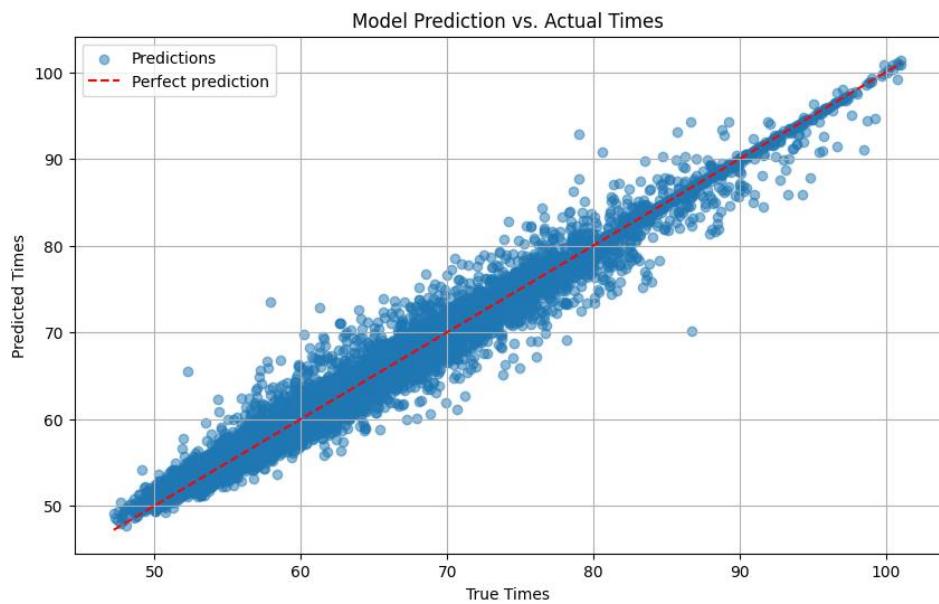


Figure 12 : Résultats modèle de Réseau de Neurones (100m nage libre)

Comme pour le modèle de régression polynomiale, la première course analysée est le 100m nage libre. Les résultats de l'entraînement de ce modèle sont extrêmement performants. Pour juger la qualité du modèle, la RMSE (Root Mean Square Error) est calculée sur les données d'entraînement, ce qui donne la qualité de l'entraînement à proprement dit, et cette même erreur est calculée sur les données de test. La comparaison des deux valeurs permet de s'assurer que le modèle n'est pas en sous- ou sur-apprentissage. Pour le résultat de l'entraînement, le modèle prédit des résultats en moyenne 1,22 seconde à côté du résultat attendu. Pour les données de test, cette erreur est même inférieure, avec 1,20 seconde. Ceci montre que le modèle n'est pas en surapprentissage. Cette petite différence est simplement due au hasard de la distribution des données d'entraînement et de test. Elle correspond à un écart de 1,88% du temps moyen.

c. Modèle final

Sur base des deux modèles appliqués et configurés, le réseau de neurones sort largement du lot en termes de performances. Le temps nécessaire à son entraînement dans la configuration actuelle est de moins de 9 minutes contre près de 27 minutes pour le modèle de régression. Ce temps est précieux, surtout dans le cas où d'autres courses doivent également être entraînées et modélisées à l'aide de la même configuration. Pour ce qui est des performances intrinsèques des modèles de prédiction, le réseau de neurones est capable de prédire des résultats 2,5 fois plus précis en moyenne. En comparant les graphiques de prédictions, la distance entre la valeur prédite et la valeur réelle du second modèle n'est jamais aussi élevée que celle du modèle de régression. Ce dernier prédit dans certains cas des temps 2, voire 3 fois supérieurs aux temps réels.

En gardant donc le modèle de réseau de neurones, l'objectif est d'appliquer cette même configuration sur les résultats des autres courses. À ce niveau, les résultats indiquent des performances de précisions assez similaires. L'écart de prédictions en pourcentage entre les prédictions obtenues et les résultats attendus oscille entre 1,74% et 2,02% du temps moyen pour chaque combinaison de distance et de style. Pour rappel, les résultats du modèle pour les 100m crawl se situent en moyenne à 1,88% du temps total. Ceci indique donc que les paramètres optimaux du réseau de neurones initialement développé sont applicables aux autres courses.

Pour approfondir l'analyse des différentes courses, certaines architectures de réseau de neurones considérées sous-optimales pour le 100m nage libre sont réévaluées ici. Voici le détail des analyses :

	Couches	Nœuds	Fonctions d'activation
Model 1	4	16 – 8 – 4 – 1	ELU
Model 2 (opti. 100m nl)	4	16 – 8 – 4 – 1	RELU
Model 3	4	4 – 8 – 4 – 1	RELU
Model 4	3	8 – 4 – 1	ELU

Courses	Modèles		model 1		model 2		model 3		model 4		Total rmse moyen		Total rmse % moyen	
			rmse moyen	rmse % moyen	rmse moyen	rmse % moyen	rmse moyen	rmse % moyen	rmse moyen	rmse % moyen	rmse moyen	rmse % moyen	rmse moyen	rmse % moyen
100m br			1,47	1,80%	1,48	1,81%	1,68	2,05%	1,49	1,82%	1,53	1,87%		
100m dos			1,31	1,79%	1,32	1,81%	1,44	1,97%	1,36	1,86%	1,3575	1,86%		
100m nl			1,22	1,89%	1,22	1,88%	1,34	2,07%	1,22	1,89%	1,25	1,93%		
100m pap			1,36	1,93%	1,42	2,02%	1,61	2,29%	1,42	2,03%	1,4525	2,07%		
200m 4n			2,57	1,64%	2,71	1,74%	2,96	1,90%	2,88	1,84%	2,78	1,78%		
200m nl			2,48	1,82%	2,47	1,82%	2,65	1,95%	2,62	1,93%	2,555	1,88%		
400 nl			5,46	1,89%	5,73	1,98%	6,7	2,32%	5,81	2,01%	5,925	2,05%		
Total général	2,267142857	1,82%	2,335714286	1,87%	2,625714286	2,08%	2,4	1,91%	2,407142857	1,92%				

Figure 13 : Performances de chaque Modèle par Course

On peut identifier 2 configurations plus précises que les autres. La seule différence entre les 2 modèles réside en l'utilisation de deux fonctions d'activation distinctes. La fonction 'ELU', bien que moins utilisée, permet une approche plus évolutive et douce lors de l'approche du seuil de la fonction, tandis que 'ReLU' réalise un changement net au niveau du seuil. La première approche semble très légèrement se démarquer en termes de performances (RMSE % moyen 1,82% contre 1,87%). Ces valeurs étant assez insignifiantes, il est intéressant de comparer la vitesse d'exécution de chacun de ces modèles. Cette information est également très importante à prendre en compte dans le cas où de nombreux nouveaux calculs sont nécessaires.

Moyenne de temps	Modèles	Course	model 1	model 2	Total général
		100m br	261,93	261,11	261,52
		100m dos	323,99	326,76	325,375
		100m nl	549,54	545,28	547,41
		100m pap	246,68	246,02	246,35
		200m 4n	268,38	262,4	265,39
		200m nl	257,48	256,46	256,97
		400 nl	216,87	240,04	228,455
		Total général	303,55	305,44	304,50

Figure 14 : Vitesse d'exécution de chaque Modèle par Course (en secondes)

À nouveau, les performances en termes de temps nécessaire à l'entraînement des modèles sont légèrement en faveur du premier modèle, pour une différence inférieure à 2 secondes. En conclusion, il est préférable de garder le modèle numéro 1, utilisant une fonction d'activation 'ELU'.

d. Prédictions

Pour réaliser les prédictions, une dernière partie de script est développée. Cette partie implique la création des informations d'une compétition que l'on souhaite prédire. Ceci implique le nom du nageur, la course réalisée et la date de cette compétition ainsi que la taille de la piscine. Sur base de ces informations, les colonnes calculées utilisées comme « features » du modèle doivent être calculées. Il faut ensuite récupérer le modèle associé à la course et utiliser la méthode `model.predict(input)`. Cette fonction donne le résultat final de la prédiction en secondes, qui est donc une estimation avec une marge d'erreur propre au modèle.

Cependant, lors de tentatives de prédictions sur différents nageurs et nageuses, une erreur surprenante s'est présentée. Régulièrement, le nageur ou la nageuse ne comportait aucun résultat, rendant la prédiction impossible. À la suite de cette constatation et d'une analyse du code et de l'évolution des données au fil de celui-ci, il est possible d'identifier que le filtre de données est trop sélectif. La distribution du nombre de nageurs utilisés lors de l'entraînement des données est extrêmement limitée. Seuls 550 à 1300 d'entre eux étaient repris dans chaque groupe de données par course. Il faut donc un retour en arrière et une modification de ce code de sélection des variables, pour augmenter le nombre de données utilisées lors de l'entraînement.

```
# Filter swimmers with > 5 races and presence at age 16
group_df = group_df.groupby('swimmer_name').filter(
    lambda x: len(x) > 5 and 16 in x['age'].values # Anciennes valeurs : min 10 courses et au moins une course à 14 et 18 ans
)
```

Figure 15 : Code de tri modifié

Grâce à ces changements, le nombre de nageurs pris en compte pour l'entraînement de chaque course a plus que triplé : entre 2500 et 4200 nageurs par course. Sur ces nouvelles données, l'application du modèle de régression polynomiale et du réseau de neurones donne des résultats différents et plus performants. Les estimations du premier modèle pour le 100m nage libre atteignent une erreur de 2,8 secondes en moyenne, contre 3,3 avant. Une prédiction du réseau de neurones ne comporte une erreur plus faible également avec 1,15 seconde (1,22 seconde avant modifications). Un nombre de données plus important implique en revanche un temps d'entraînement plus long. Le modèle de régression nécessite près de 2 heures pour atteindre une forme optimale, soit plus de 4 fois plus long. Il faut un peu moins de 23 minutes pour l'entraînement du réseau de neurones, contre 9 minutes avant la modification. Ceci justifie encore davantage l'utilisation de cette dernière approche pour créer des modèles adaptés à toutes les autres courses.

```
Using model for 100_Freestyle: 100_Freestyle
1/1 ━━━━━━━━━━━━━━━━ 0s 34ms/step
 Predicted time for Gjon VATA in 100m Freestyle on 2025-05-27:
53.96 seconds
```

X. Visualisation

Une partie des objectifs du projet est de proposer des dashboards à l'aide d'un outil BI. L'un doit permettre aux clubs de pouvoir voir les tendances d'évolution de tous les nageurs. Le second offre aux nageurs une possibilité de se comparer avec les résultats d'autres nageurs. Pour réaliser ces dashboards, l'application Power BI a été utilisée pour la gratuité d'une partie de son utilisation et la personnalisation qu'elle offre. Sa connectivité avec des bases de données afin d'avoir accès à des données dynamiques est un atout supplémentaire.

a. Clubs

L'outil proposé aux clubs permet d'avoir une vue d'ensemble sur les résultats des nageurs et nageuses du club. Il montre principalement la distribution des résultats en fonction de divers facteurs comme l'âge, la course ou l'année de naissance. Ces distributions permettent de voir l'évolution de la taille du club et de gérer le flux de nageurs. D'autres graphes permettent d'identifier l'évolution des résultats moyens et individuels par âge. L'interactivité de ce dashboard est importante. Beaucoup de listes et de 'segments' sont présentés pour permettre de comparer des données plus précises, comme identifier la courbe d'évolution d'un nageur spécifique ou identifier la course la plus nagée au cours d'une année précise. De plus, l'interactivité entre les graphes est gérée de telle manière à pouvoir garder un aperçu global de certaines informations tout en étant capable de se focaliser sur un élément spécifique. Beaucoup de couleurs et d'informations sont présentes, mais les informations clés restent facilement identifiables et compréhensibles.

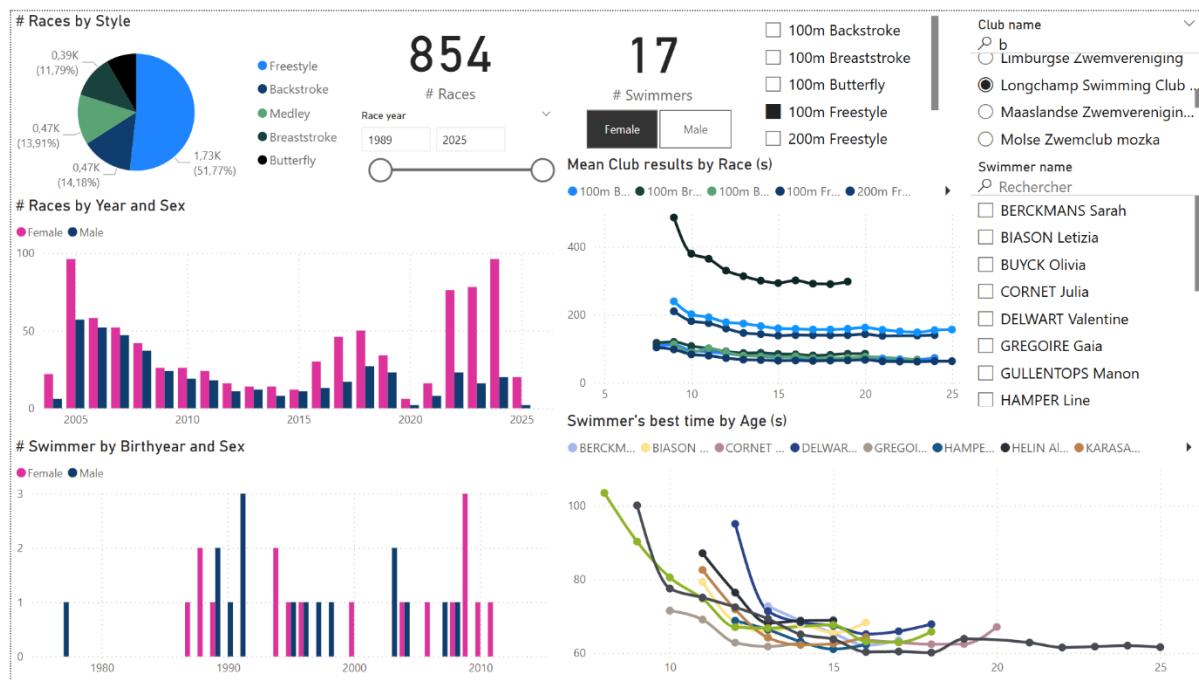


Figure 16 : Dashboard dédié aux Clubs de natation

Grâce à ce dashboard, un entraîneur peut déjà identifier le potentiel de certains nageurs. La direction d'un club peut également utiliser cet outil pour avoir un aperçu de l'évolution et de l'état actuel du club. L'exemple ci-dessus démontre une croissance forte du nombre de courses depuis 2016. La diminution en 2020 et 2021 est liée au Covid et n'est donc pas représentative du statut du club.

b. Nageurs

Le second rapport visuel Power BI destiné aux nageurs, mais également pertinent pour les entraîneurs, offre la possibilité de voir l'évolution d'un nageur et de comparer celle-ci avec un ou plusieurs autres nageurs. Il permet de présenter et comparer les meilleurs temps et le nombre de participation à chaque course. De plus, une carte est ajoutée pour identifier les endroits où les résultats ont été réalisés. Pour finir, les courbes d'évolution du record personnel à chaque âge sont présentées. Ce visuel comporte également des segments de tri afin de pouvoir sélectionner le ou les nageurs à comparer, ainsi que la course dont la courbe d'évolution est présentée.



Figure 17 : Dashboard dédié aux Nageurs

L'exemple affiché ci-dessus est une comparaison entre 2 nageurs, a priori frères. Le premier graphe montre que 'Ruben' a des résultats plus rapides pour toutes les courses sauf les 100m Backstroke (dos) et 100m Breaststroke (brasse). Les meilleurs temps sur le 100m Freestyle (nage libre) sont très proches. De plus, cette dernière course est celle dont le nombre de participations est le plus élevé comme l'indique le graphe du nombre de courses par nageur. La carte des lieux de compétitions indique que la plupart des résultats pour le 100m nage libre ont été réalisés à Anvers. Grâce au dernier visuel, présentant l'évolution des temps avec l'âge, l'identification du nageur le plus rapide sur la course sélectionnée est réalisable facilement.

XI. Conclusion

Ce travail fut l'occasion d'appliquer un grand nombre d'acquis des cours du Bachelier de Spécialisation. Chaque objectif fixé au début du projet a été abordé, analysé et implémenté. La réflexion et l'analyse des résultats sur base de ces objectifs est encore nécessaire. Pour finaliser un projet, il faut également expliquer comment le projet doit être délivré et utilisé. Tout projet comporte également une réflexion globale sur les pistes d'amélioration possibles et une analyse sur les difficultés rencontrées.

a. Evaluation

Le premier objectif du projet était de réfléchir et de modéliser une base de données. Cette étape cruciale pour le reste du projet offre un résultat satisfaisant. Toutes les données pertinentes sont reprises au sein d'une structure normalisée. Cette partie n'a pas été possible sans la mise en place d'un flux ETL. Celui-ci a permis d'avoir des données complètes et structurées indispensables pour la suite du projet.

Les dashboards Power BI réalisés respectent également les objectifs initialement prévus. Chaque cible est capable d'identifier les informations clés afin de comprendre et d'analyser les performances. La seule information non reprise sur ces visuels est le résultat des prédictions.

Ces résultats, obtenus à l'aide des modèles de data mining, offrent des résultats satisfaisants. Les prédictions estiment une valeur seulement en moyenne 1,71% supérieure ou inférieure au temps total de la course. Cette précision permet de prédire des résultats à très court terme (<6 mois – 1 an). Étant donné que le modèle repose sur des variables temporelles, il est plus compliqué de réaliser ses prédictions sur le long terme. Pour estimer un résultat longtemps à l'avance, il faudrait estimer des résultats intermédiaires avec pour chacun d'entre eux une marge d'erreur, qui s'accumule et mène à une marge trop importante pour que le résultat soit représentatif.

b. Pistes d'amélioration

Le projet dans sa globalité est complet et fonctionnel. Il y a cependant toujours des points qui pourraient mériter une attention plus particulière, ou des fonctionnalités supplémentaires qui permettraient d'améliorer le projet final.

Une des améliorations qui pourrait améliorer le projet final est de travailler sur une gestion de la cohérence des informations de lieux de compétition. Ces données ne peuvent être utilisées en l'état pour tirer des conclusions correctes. Pour cela, il serait possible d'appliquer des règles générales ou d'appliquer des techniques de correspondances automatiques à l'aide de librairies Python pour réaliser du « fuzzy matching ».

Au niveau des données, il pourrait également être pertinent de récupérer davantage de résultats. Certaines courses n'ont pas été analysées et n'ont pas de modèle de prédiction adapté. Si un nageur est particulièrement fort sur une seule distance, comme le 1500m nage libre, il est impossible d'estimer des résultats, ni de comparer ces performances avec d'autres sportifs.

Une dernière amélioration, qui est nécessaire pour pouvoir faire évoluer le modèle et avoir des estimations réalistes sur le long terme, est d'ajouter un script capable de récupérer les résultats des nouvelles compétitions, et ensuite de réentraîner les modèles. Comme expliqué plus tôt, ce modèle est très dépendant de données temporelles, et le manque de données récentes est le facteur principal de la diminution de la pertinence des prédictions. Cela pourrait être réalisé avec davantage de temps à l'aide des ressources et des résultats publiés chaque semaine sur swimrankings.net.

c. Difficultés rencontrées

Un travail aussi large implique également des problèmes et des difficultés. Dès le début, le défi pour récupérer les données est le temps nécessaire. La méthode de collecte des résultats repose sur le Web Scraping. C'est un processus long qui a nécessité une réflexion approfondie pour réduire le nombre de données finalement récupérées. Malgré la réduction du nombre de nageurs total à 6651 nageurs sur les presque 20.000 athlètes avec un résultat officiel, et la décision de ne garder que les 7 courses les plus pertinentes, les 2 serveurs étaient réellement nécessaires pour optimiser le temps de collecte.

La seconde difficulté fut rencontrée lors de la création des visuels sur Power BI. En plus du souci lié aux villes et pays, qui rend l'analyse des données impossible, les outils de BI ne sont pas bien adaptés pour afficher des données de type durée. La solution est d'utiliser deux colonnes distinctes pour la même valeur. L'une au format nombre décimal, qui donne le temps en secondes (valeur récupérée dans la base de données), et l'autre au format texte, qui extrait les minutes de la valeur numérique et modifie la mise en forme du texte au format 'mm:ss.ff'. Cette valeur peut ensuite être affichée comme étiquette ou dans la bulle d'information d'une valeur sur un graphe.

Pour finir, optimiser et entraîner plusieurs modèles nécessite beaucoup de temps. Malgré les connaissances théoriques acquises où cours de la formation, il n'est pas évident d'implémenter cela. Mes compétences en informatique et en développement Python m'ont beaucoup aidé lors de cette phase. Finalement, le résultat obtenu est extrêmement satisfaisant, et avoir pu appliquer ces techniques moi-même est une expérience supplémentaire importante pour le développement futur.

XII. Bibliographie

1. **Belgium results**, Swimrankings, <https://www.swimrankings.net/index.php?page=rankingDetail&club=BEL>, Consulté dernièrement le 20 mai 2025
2. **Swimmer results**, Swimrankings, <https://www.swimrankings.net/index.php?page=athleteDetail&athleteId=XXXXXX>, Consulté dernièrement le 20 mai 2025
3. **FINA Swimming points**, World Aquatics, <https://www.worldaquatics.com/swimming/points>, Consulté le 23 mai 2025
4. **Réseau de neurones en Python**, eaQbe, https://docs.eaqbe.com/fr/machine_learning/neural_network, Consulté dernièrement le 10 avril 2025
5. **Documentation SSIS**, Microsoft, <https://learn.microsoft.com/fr-fr/sql/integration-services/sql-server-integration-services?view=sql-server-ver17>, Consulté dernièrement en février 2025
6. **Stackoverflow**, Stackoverflow, <https://stackoverflow.com/>, Consulté dernièrement en avril 2025

Usage de l'IA générative

Chat GPT : <https://chatgpt.com/share/6838db78-1cf8-8006-97f4-0219c6c0adf9>

Annexes

Tous les documents créés et utiles au projet sont trouvables à l'adresse suivante : <https://github.com/Swithan/Swimstats>. Les documents principaux sont repris dans les annexes ci-dessous.

Annexes	I
I. Model.ipynb, Code python	II
II. Swimstats.sql, tables de base de données	XII
III. Récupération des données.....	XIV
a. swim_database.py, requêtes SQL	XIV
b. s.py, code du scraping	XVIII
c. Dockerfile.db, image Docker SQL	XXIII
d. Dockerfile.py, image Docker Python	XXIII

I. Model.ipynb, Code python

```
Import libraries and data
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, ELU
from sklearn.metrics import mean_squared_error
import time
import matplotlib.pyplot as plt
import seaborn as sns
import math, copy

# Create results csv with SQL Query if needed
data = pd.read_csv('results.csv')

1. Data Understanding
# Combine distance_metre and style_name to create a unique race
identifier
data['race'] = data['distance_metre'].astype(str) + 'm ' +
data['style_name']

# Define the order for pool_metre for consistent coloring
pool_order = ['25m', '50m']

blue_palette = sns.color_palette("Blues", n_colors=len(pool_order))

# Chart 1: Distribution of data by race and pool size
plt.figure(figsize=(15, 8))
sns.boxplot(x='race', y='time_millisecond', hue='pool_metre',
data=data, palette=blue_palette, order=data['race'].unique())
plt.title('Distribution of Time by Race and Pool Size (All Data)')
plt.xlabel('Race')
plt.ylabel('Time (s)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

data_filtered = pd.DataFrame()

# filter on 99th quantile
for race_name, race_group in data.groupby('race'):
    time_threshold = race_group['time_millisecond'].quantile(0.99)

    filtered_group = race_group[race_group['time_millisecond'] <
time_threshold]
```

```

data_filtered = pd.concat([data_filtered, filtered_group])

# Chart 2: Distribution of data by race and pool size, removing
extreme values
plt.figure(figsize=(15, 8))
sns.boxplot(x='race', y='time_millisecond', hue='pool_metre',
data=data_filtered, palette=blue_palette,
order=data_filtered['race'].unique())
plt.title('Distribution of Time by Race and Pool Size (Filtered
Data)')
plt.xlabel('Race')
plt.ylabel('Time (s)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

o_data = data
data = data_filtered
2. Data Preparation
# Convert race_date to datetime
data['race_date'] = pd.to_datetime(data['race_date'])

# Compute season year (swimming convention)
data['month'] = data['race_date'].dt.month
data['year'] = data['race_date'].dt.year
data['season_year'] = data['year']
data.loc[data['month'] <= 8, 'season_year'] -= 1

# Compute age (season_year - birth_year)
data['birth_year'] = pd.to_datetime(data['birth_year'],
errors='coerce')
data['age'] = data['season_year'] - data['birth_year'].dt.year

# Normalize column names
data['time'] = data['time_millisecond']
data['distance'] = data['distance_metre']
data['stroke'] = data['style_name']
data['pool'] = data['pool_metre'].apply(lambda x: '50m' if x == 50
else '25m')

# Drop rows with missing NaN/empty values
data = data.dropna(subset=['swimmer_name', 'race_date', 'time',
'age'])

# Define the function to compute a trend score
def compute_trend(group, window=3):
    trends = []
    for i in range(len(group)):
        if i < window - 1:
            trends.append(np.nan)
        else:

```

```

        y = group['time'].iloc[i - window + 1:i +
1].values.reshape(-1, 1)
        x = np.arange(window).reshape(-1, 1)
        model = LinearRegression().fit(x, y)
        slope = model.coef_[0][0]
        trends.append(slope)
    return pd.Series(trends, index=group.index)

data = pd.get_dummies(data, columns=['sex', 'pool'], dtype='int')

# Process each unique race event
group_cols = ['distance', 'stroke']
event_dfs = {}

for group_key, group_df in data.groupby(group_cols):
    print(f"Processing event: {group_key}")
    group_df = group_df.sort_values(by=['swimmer_name',
'race_date']).copy()

    # Previous race time per swimmer
    group_df['previous_time'] =
group_df.groupby('swimmer_name')['time'].shift(1)
    group_df.dropna(subset=['previous_time'], inplace=True)

    # Previous race date per swimmer
    group_df['previous_race'] =
group_df.groupby('swimmer_name')['race_date'].shift(1)
    group_df['days_since_last_race'] = (group_df['race_date'] -
group_df['previous_race']).dt.days

    # Trend score
    group_df['trend_score'] = group_df.groupby('swimmer_name',
group_keys=False).apply(lambda g: compute_trend(g, window=3))

    # Age at first race per swimmer
    group_df['first_race_age'] =
group_df.groupby('swimmer_name')['age'].transform('first')

    group_df = group_df.dropna()

    # Filter swimmers with > 5 races and presence at age 16
    group_df = group_df.groupby('swimmer_name').filter(
        lambda x: len(x) > 5 and 16 in x['age'].values # Anciennes
valeurs : min 10 courses et au moins une course a 14 et 18 ans
    )

    if not group_df.empty:
        event_dfs[group_key] = group_df

print(f"Processed {len(event_dfs)} events.")
2.1. Visualise Changes

```

```

# Convert to list of (key, df) items
for (distance, stroke), df in event_dfs.items():
    # print(f"Event: {distance}m {stroke} - Number of swimmers:
{df['swimmer_name'].nunique() }")
    if (stroke == 'Freestyle') & (distance == 100):
        # Compute y-axis limits with 5% padding
        times = df['time'].tolist()
        y_min = min(times)
        y_max = max(times)
        y_pad = (y_max - y_min) * 0.05
        y_min -= y_pad
        y_max += y_pad

        # Plot
        plt.figure(figsize=(8, 6))
        plt.plot(df['age'], df['time'], 'o', alpha=0.6)
        plt.xlabel("Age")
        plt.ylabel("Time (s)")
        plt.title(f"{distance}m {stroke}")
        plt.ylim(y_min, y_max)
        plt.grid(True)
        plt.tight_layout()
        plt.show()

3. Data Modeling
    a) Logistic Regression
data_pr = event_dfs[(np.int64(100), 'Freestyle')].copy()

x = data_pr[['age', 'sex_M', 'pool_50m', 'previous_time',
'days_since_last_race', 'trend_score', 'first_race_age']].values
y = data_pr['time'].values

poly = PolynomialFeatures(degree=2)

scaler_x = StandardScaler()
scaler_y = StandardScaler()

x_poly = poly.fit_transform(x)

y = scaler_y.fit_transform(y.reshape(-1, 1))
x_poly = scaler_x.fit_transform(x_poly)

x_train, x_test, y_train, y_test = train_test_split(x_poly, y,
test_size=0.4, random_state=42)

def predict(x, w, b):
    p = np.dot(x, w) + b
    return p

def compute_cost(X, y, w, b, lambda_):
    m = X.shape[0]
    cost = 0.0
    for i in range(m):

```

```

f_wb_i = np.dot(X[i], w) + b
cost = cost + (f_wb_i - y[i])**2
cost = cost / (2 * m)

# regularisation L2 (ridge)
regularization_cost = (lambda_ / (2 * m)) * np.sum(w**2)
cost += regularization_cost
return cost

def compute_gradient(X, y, w, b, lambda_):
    m, n = X.shape
    dj_dw = np.zeros((n,))
    dj_db = 0.

    for i in range(m):
        err = (np.dot(X[i], w) + b) - y[i]
        for j in range(n):
            dj_dw[j] = dj_dw[j] + err * X[i, j]
        dj_db = dj_db + err
    dj_dw = dj_dw / m
    dj_db = dj_db / m
    dj_dw += (lambda_ / m) * w

    return dj_db, dj_dw

def gradient_descent(X, y, w_in, b_in, cost_function,
gradient_function, alpha, num_iters, lambda_):
    J_history = []
    w = copy.deepcopy(w_in)
    b = b_in

    for i in range(num_iters):
        dj_db, dj_dw = gradient_function(X, y, w, b, lambda_)
        w = w - alpha * dj_dw
        b = b - alpha * dj_db

        if i<100000:
            J_history.append( cost_function(X, y, w, b, lambda_))

        if i > 1 and abs(J_history[-1] - J_history[-2]) < 0.0001:
            print(f"Early stopping at iteration {i} as cost change
is less than 0.0001")
            break

        if i% math.ceil(num_iters / 10) == 0:
            print(f"Iteration {i:4}: Cost {J_history[-1]} ",
                  f"dj_dw: {dj_dw}, dj_db: {dj_db} ",
                  f"w: {w}, b:{b}")

    return w, b, J_history

initial_w = np.random.randn(x_train.shape[1])

```

```

initial_b = 0

lambda_ = 0.1
iterations = 1000
alpha = 0.05
w_final, b_final, J_hist = gradient_descent(x_train, y_train,
initial_w, initial_b,
compute_cost,
compute_gradient,
alpha,
iterations, lambda_)

print(f"(w,b) found by gradient descent: ({w_final},{b_final})")

def predict(X, w, b):

    return np.dot(X, w) + b

def rmse(y_true, y_pred):
    mse = np.mean((y_true - y_pred) ** 2)
    return np.sqrt(mse)

pr_y_pred = predict(x_test, w_final, b_final)
pr_y_test = y_test

pr_y_pred_original = scaler_y.inverse_transform(pr_y_pred.reshape(-1, 1))
pr_y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1))

pr_rmse_value_original = rmse(pr_y_test_original,
pr_y_pred_original)

pr_rmse_percentage = (pr_rmse_value_original /
np.mean(pr_y_test_original)) * 100

print(f"RMSE on the test set (original scale):"
{pr_rmse_value_original}")
print(f"RMSE en pourcentage sur l'ensemble de test:"
{pr_rmse_percentage:.2f}%)

pr_y_test_true = scaler_y.inverse_transform(pr_y_test)
pr_y_test_pred = scaler_y.inverse_transform(pr_y_pred.reshape(-1, 1))

# Création du graphique
plt.figure(figsize=(10, 6))
plt.scatter(pr_y_test_true, pr_y_test_pred, alpha=0.6, color='blue',
label='Prédictions')
plt.plot([pr_y_test_true.min(), pr_y_test_true.max()],
[pr_y_test_true.min(), pr_y_test_true.max()],
'r--', label='Ligne parfaite (y = x)')

```

```

plt.xlabel("True Times (seconds)")
plt.ylabel("Estimated Times (seconds)")
plt.title("Neural network predictions vs true times\n 100m\nFreestyle")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Select features and target
features = ['swimmer_num_id', 'age', 'sex_M', 'previous_time',
'pool_50m', 'days_since_last_race', 'trend_score', 'first_race_age']
target = 'time'

models = {}
scalers = {}

# run for each race
for (distance, stroke), df in event_dfs.items():

    X = df[features]
    y = df[target]

    # Split the dfs
    x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Scale the dfs
    scaler = StandardScaler()
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)
    scalers[f'{distance}_{stroke}'] = scaler

    # define models to train
    def build_models():
        tf.random.set_seed(20)

        model_1 = Sequential([
            Dense(16, activation=ELU(alpha=1.0),
input_shape=(x_train.shape[1],)),
            Dense(8, activation=ELU(alpha=1.0)),
            Dense(4, activation=ELU(alpha=1.0)),
            Dense(1, activation='linear')
        ], name='model_1')

        model_2 = Sequential([
            Dense(16, activation='relu',
input_shape=(x_train.shape[1],)),
            Dense(8, activation='relu'),
            Dense(4, activation='relu'),
            Dense(1, activation='linear')
        ], name='model_2')


```

```

model_3 = Sequential([
    Dense(4, activation='relu',
input_shape=(x_train.shape[1],)),
    Dense(8, activation='relu'),
    Dense(4, activation='relu'),
    Dense(1, activation='linear')
], name='model_3')

model_4 = Sequential([
    Dense(8, activation=ELU(alpha=1.0),
input_shape=(x_train.shape[1],)),
    Dense(4, activation=ELU(alpha=1.0)),
    Dense(1, activation='linear')
], name='model_4')

# model_list = [model_1, model_2, model_3, model_4]

model_list = [model_1]

return model_list

nn_models = build_models()
print(f"Training {distance} {stroke}...")

# Train each model
for model in nn_models:
    start_time = time.time()
    model.compile(
        loss='mse',
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)
    )

    model._name = f"{distance}_{stroke}".replace(" ", "_")
    model.name = f"{distance}_{stroke}".replace(" ", "_")

    model.fit(
        x_train, y_train,
        epochs=500,
        batch_size=64,
        validation_split=0.1,
        verbose=0
    )

yhat_train = model.predict(x_train)
yhat_test = model.predict(x_test)

# Check for NaNs in predictions
if np.isnan(yhat_train).any() or np.isnan(yhat_test).any():

```

```

        print(f"Model {distance} {stroke} - Predictions contain
NaNs")
    continue

    rmse_train = np.sqrt(mean_squared_error(y_train,
yhat_train))
    rmse_test = np.sqrt(mean_squared_error(y_test, yhat_test))

    rmse_percentage = (rmse_test / np.mean(y_test)) * 100

    print(f"Model {model.name} - RMSE on Training Set:
{rmse_train:.4f}")
    print(f"Model {model.name} - RMSE on Test Set:
{rmse_test:.4f}\n")
    print(f"Model {model.name} - RMSE Percentage on Test Set:
{rmse_percentage:.2f}%\n")

    print(f"Training time: {time.time() - start_time:.2f}
seconds\n")

    # Save the model
    models[f'{distance}_{stroke}'] = model

    # Plot predictions vs true times
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, yhat_test, alpha=0.5,
label="Predictions")
    plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'r--', label="Perfect prediction")
    plt.xlabel("True Times")
    plt.ylabel("Predicted Times")
    plt.title("Neural network predictions vs true times\n" +
f"{distance}m {stroke}")
    plt.legend()
    plt.grid(True)
    plt.show()

Predictions
# New swimmer competition to predict
new_results = {
    "swimmer": 'Gjon VATA',
    "race_date": '2025-05-27',
    "distance": 100,
    "stroke": 'Freestyle',
    "pool": '50m'
}

# Retrieve swimmer history for that event
event_key = (new_results['distance'], new_results['stroke'])
event_df = event_dfs.get(event_key)

if event_df is None:
    raise ValueError(f"No event data for {event_key}")

```

```

swimmer_history = event_df[event_df['swimmer_name'] == new_results['swimmer']].copy()
if swimmer_history.empty:
    raise ValueError(f"No history found for swimmer {new_results['swimmer']}")

swimmer_history = swimmer_history.sort_values(by='race_date',
                                             ascending=False)

# Features from swimmer history
new_results["swimmer_num_id"] =
    swimmer_history['swimmer_num_id'].iloc[0]
new_results["birth_year"] =
    pd.to_datetime(swimmer_history['birth_year'].iloc[0],
                  errors='coerce').year
new_results["sex_M"] = swimmer_history['sex_M'].iloc[0]
new_results["first_race_age"] =
    swimmer_history['first_race_age'].iloc[0]
new_results['race_date'] = pd.to_datetime(new_results['race_date'])
new_results["pool_50m"] = 1 if new_results["pool"] == "50m" else 0

# Compute seasonal age for new prediction
month = new_results['race_date'].month
year = new_results['race_date'].year
season_year = year - 1 if month <= 8 else year
new_results['age'] = season_year - new_results['birth_year']

# Get last race info (same swimmer, same distance, same stroke,
# before current race)
past_races = event_df[
    (event_df['swimmer_name'] == new_results['swimmer']) &
    (event_df['race_date'] < new_results['race_date'])].sort_values(by='race_date', ascending=False)

if not past_races.empty:
    last_race = past_races.iloc[0]
    new_results["previous_time"] = last_race['time']
    new_results["days_since_last_race"] = (new_results['race_date'] -
                                           last_race['race_date']).days
    new_results["trend_score"] = last_race['trend_score']
else:
    new_results["previous_time"] = swimmer_history['time'].iloc[-1]
    new_results["days_since_last_race"] = 9999
    new_results["trend_score"] = 0

# Prepare the input for prediction
features = ['swimmer_num_id', 'age', 'sex_M', 'previous_time',
            'pool_50m',
            'days_since_last_race', 'trend_score', 'first_race_age']

```

```

input_df = pd.DataFrame([new_results])[features]

# Scaling
scaler_key =
f"{new_results['distance']}_{new_results['stroke']}".replace(" ", "_")
scaler = scalers.get(scaler_key)
if scaler is None:
    raise ValueError(f"No scaler found for {scaler_key}")

input_scaled = scaler.transform(input_df)

# Prediction
model = models.get(scaler_key)
print(f"Using model for {scaler_key}: {model.name if model else 'None'}")
if model is None:
    raise ValueError(f"No model found for {scaler_key}")

prediction = model.predict(input_scaled)
predicted_time = prediction
# Output
print(f"☑ Predicted time for {new_results['swimmer']} in
{new_results['distance']}m {new_results['stroke']} on
{new_results['race_date'].date()} : {predicted_time[0][0]:.2f} seconds")

```

II. Swimstats.sql, tables de base de données

```

/* Snowflake */
CREATE TABLE IF NOT EXISTS swimstats.club (
    club_num_id SERIAL PRIMARY KEY,
    club_name domaine.nom NOT NULL
) ;

CREATE TABLE IF NOT EXISTS swimstats.style (
    style_num_id SERIAL PRIMARY KEY,
    style_name domaine.nom NOT NULL
) ;

CREATE TABLE IF NOT EXISTS swimstats.distance (
    distance_num_id SERIAL PRIMARY KEY,
    distance_metre domaine.metre NOT NULL
) ;

CREATE TABLE IF NOT EXISTS swimstats.sex (
    sex_code domaine.code PRIMARY KEY,
    sex_name domaine.nom NOT NULL
) ;

CREATE TABLE IF NOT EXISTS swimstats.country (

```

```

country_code domaine.code PRIMARY KEY,
country_name domaine.nom NOT NULL
) ;

/* Star */
CREATE TABLE IF NOT EXISTS swimstats.place (
    place_num_id SERIAL PRIMARY KEY,
    city_name domaine.nom NOT NULL,
    country_code domaine.nom NOT NULL REFERENCES
swimstats.country(country_code)
) ;

CREATE TABLE IF NOT EXISTS swimstats.race (
    race_num_id SERIAL PRIMARY KEY,
    distance_num_id INT NOT NULL REFERENCES
swimstats.distance(distance_num_id),
    style_num_id INT NOT NULL REFERENCES
swimstats.style(style_num_id)
) ;

CREATE TABLE IF NOT EXISTS swimstats.swimmer (
    swimmer_num_id domaine.num_id PRIMARY KEY,
    swimmer_firstname domaine.prenom NOT NULL,
    swimmer_lastname domaine.nom NOT NULL,
    birth_year domaine.annee NOT NULL,
    club_num_id domaine.num_id NOT NULL REFERENCES
swimstats.club(club_num_id),
    sex_code domaine.code NOT NULL REFERENCES
swimstats.sex(sex_code),
    nationality_code domaine.code NOT NULL REFERENCES
swimstats.nationality(nationality_code)
) ;

/* Fact */

CREATE TABLE IF NOT EXISTS swimstats.result (
    result_num_id SERIAL PRIMARY KEY,
    swimmer_num_id domaine.num_id NOT NULL REFERENCES
swimstats.swimmer(swimmer_num_id),
    race_num_id domaine.num_id NOT NULL REFERENCES
swimstats.race(race_num_id),
    place_num_id domaine.num_id NOT NULL REFERENCES
swimstats.place(place_num_id),
    pool_metre domaine.metre NOT NULL,
    race_date domaine.debut_date NOT NULL,
    time_millisecond domaine.duree_seconde NOT NULL,
    race_points domaine.cote NOT NULL
) ;

```

III. Récupération des données

a. swim_database.py, requêtes SQL

```
import psycopg2
from psycopg2 import sql

# Step 1: Connect to the PostgreSQL database
def connect_to_db():
    try:
        conn = psycopg2.connect(
            dbname="swimstats",
            user="postgres",
            password="admin",
            # host="172.18.0.2",    # py
            host="172.19.0.2",    # pyf
            port="5432"
        )
        return conn
    except Exception as e:
        print(f"Error connecting to the database: {e}")
        return None

# Step 2: Create tables
def create_tables(conn):
    try:
        with conn.cursor() as cur:
            # Create Swimmer table
            cur.execute('''
                CREATE TABLE IF NOT EXISTS Swimmers (
                    s_id VARCHAR(255) PRIMARY KEY, -- Athlete ID
                    (unique)
                    lastname VARCHAR(255) NOT NULL,
                    firstname VARCHAR(255) NOT NULL,
                    birthyear INTEGER,
                    sex VARCHAR(1),
                    nationality VARCHAR(50),
                    last_club VARCHAR(255),
                    pb VARCHAR(50),
                    results BOOLEAN DEFAULT FALSE
                );
            ''')

            # Create Race table
            cur.execute('''
                CREATE TABLE IF NOT EXISTS Races (
                    r_id SERIAL PRIMARY KEY, -- Automatically
                    generated race ID
                    s_id VARCHAR(255) REFERENCES Swimmers(s_id) ON
                    DELETE CASCADE,
                    distance VARCHAR(10) NOT NULL,
            ''')
```

```

        stroke VARCHAR(50),
        pool VARCHAR(50),
        time VARCHAR(50),
        points INTEGER,
        race_date DATE,
        city VARCHAR(255),
        country VARCHAR(255)
    );
''')

# Create splits table
cur.execute('''
    CREATE TABLE IF NOT EXISTS Race_Splits (
        rs_id SERIAL PRIMARY KEY,
        race_id INTEGER REFERENCES Races(r_id) ON DELETE
CASCADE,
        split_distance VARCHAR(10),
        split_time VARCHAR(50),
        subsplit_distance VARCHAR(10),
        subsplit_time VARCHAR(50)
    );
''')


conn.commit()
print("Tables created successfully.")
with (open('swimmers.csv', 'r')) as f:
    cur.copy_expert("COPY Swimmers FROM STDIN WITH CSV
HEADER", f)
    conn.commit()
except Exception as e:
    print(f"Error creating tables: {e}")
    conn.rollback()

def insert_swimmer(conn, swimmer_data):
    try:
        with conn.cursor() as cur:
            # Insert swimmer data into the Swimmer table
            cur.execute('''
                INSERT INTO Swimmers (s_id, lastname, firstname,
                birthyear, sex, nationality, last_club, pb)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
                ON CONFLICT (s_id) DO NOTHING;
            ''', (swimmer_data['s_id'], swimmer_data['lastname'],
            swimmer_data['firstname'],
            swimmer_data['birthyear'], swimmer_data['sex'],
            swimmer_data['nationality'], swimmer_data['last_club'],
            swimmer_data['pb']))
            conn.commit()
    except Exception as e:
        print(f"Error inserting swimmer: {e}")
        conn.rollback()

```

```

def insert_race(conn, race_data):
    try:
        with conn.cursor() as cur:
            # Insert race data into the Race table
            cur.execute('''
                INSERT INTO Races (s_id, distance, stroke, pool,
time, points, race_date, place)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
                RETURNING r_id;
            ''', (race_data['s_id'], race_data['distance'],
race_data['stroke'], race_data['pool'],
            race_data['time'], race_data['points'],
race_data['race_date'], race_data['place']))

            inserted_r_id = cur.fetchone()[0]
            conn.commit()
            return inserted_r_id
    except Exception as e:
        print(f"Error inserting race: {e}")
        conn.rollback()
        return None

def get_races(conn):
    try:
        with conn.cursor() as cur :
            cur.execute('''SELECT * FROM races''')
            return cur.fetchall()
    except Exception as e:
        print(f"Error retrieving races : {e}")
        conn.rollback()

def insert_split(conn, r_id, split_data):
    try:
        with conn.cursor() as cur:
            # Insert race data into the Race table
            cur.execute('''
                INSERT INTO race_splits (race_id, split_distance,
split_time, subsplit_distance, subsplit_time)
                VALUES (%s, %s, %s, %s, %s);
            ''', (r_id, split_data['split_dist'],
split_data['split_time'],
            split_data['subsplit_dist'],
split_data['subsplit_time']))

            conn.commit()
    except Exception as e:
        print(f"Error inserting split: {e}")
        conn.rollback()

```

```

def get_todo_swimmers(conn):
    try:
        with conn.cursor() as cur :
            cur.execute('''SELECT s_id FROM swimmers WHERE results = False;''')
            return cur.fetchall()
    except Exception as e:
        print(f"Error retreiving TODO swimmers : {e}")
        conn.rollback()

def get_f_todo_swimmers(conn):
    try:
        with conn.cursor() as cur :
            cur.execute('''SELECT s_id FROM swimmers WHERE results = False AND sex = 'F';''')
            return cur.fetchall()
    except Exception as e:
        print(f"Error retreiving TODO swimmers : {e}")
        conn.rollback()

def get_done_swimmers(conn):
    try:
        with conn.cursor() as cur :
            cur.execute('''SELECT s_id, firstname, lastname FROM swimmers WHERE results;''')
            return cur.fetchall()
    except Exception as e:
        print(f"Error retreiving TODO swimmers : {e}")
        conn.rollback()

def get_swimmer_by_club(conn, club):
    try:
        with conn.cursor() as cur :
            cur.execute('''SELECT s_id FROM swimmers HAVING last_club = ;''')
            return cur.fetchall()
    except Exception as e:
        print(f"Error retreiving swimmers by club : {e}")
        conn.rollback()

def update_swimmer_has_results(conn, swimmer):
    try:
        with conn.cursor() as cur :
            cur.execute('''UPDATE swimmers SET results = true WHERE s_id = %s;''', (swimmer,))
            updated_row_count = cur.lastrowid
            return updated_row_count > 0
    except Exception as e:
        print(f"Error updating done swimmers : {e}")
        conn.rollback()

```

b.s.py, code du scraping

```
import requests
from bs4 import BeautifulSoup
import time
import random
from fake_useragent import UserAgent

from swim_database import connect_to_db, create_tables,
get_done_swimmers, get_races, insert_swimmer, insert_race,
get_f_todo_swimmers, get_todo_swimmers, insert_split,
update_swimmer_has_results

def retrieve_swimmers(url, start, end, increment):
    all_results = []

    while start < end:

        ua = UserAgent()
        headers = {'User-Agent': ua.random}
        # Construct the URL for each page
        call = url + str(start)

        start_time = time.time()
        # Make a request to the webpage
        response = requests.get(call, headers=headers)

        response_time = time.time() - start_time

        # Check if the request was successful
        if response_time > 10:
            print(f"Failed to retrieve page at {call}")
            print(f"Last working id is {start}")
            break

        # Parse the page content with BeautifulSoup
        soup = BeautifulSoup(response.text, 'html.parser')

        # Extract the relevant data
        table = soup.find("table", class_="rankingList") # Example
selector

        results = table.find_all('tr', class_=['rankingList0',
'rankingList1'])

        # If no more results, break the loop
        if not results:
            break

        # Store the results
        for result in results:
```

```

        cells = result.find_all('td')

        athlete_cell = cells[0] # The first <td> contains the
        athlete's name and link
        athlete_name = athlete_cell.text.strip()

        # Find the <a> tag and extract the 'athleteId' from the
        href
        athlete_link = athlete_cell.find('a')
        athlete_id = None
        if athlete_link and 'href' in athlete_link.attrs:
            href = athlete_link['href']
            athlete_id = href.split('athleteId=')[1]

        # Extract the rest of the data (excluding the athlete's
        name)
        other_data = [cell.text.strip() for cell in cells[1:9]]
        # Extract all data except name, date, and city

        # Step 7: Extract the date (10th <td> cell) and clean it
        (without formatting)
        date_text = cells[9].text.strip() # Get the raw date
        text
        date_text_clean = date_text.replace('\xa0', ' ') # Replace non-breaking spaces

        # Step 8: Extract the city (11th <td> cell)
        city = cells[10].text.strip()

        # Combine athlete's name, athleteId, other data, cleaned
        date, and city
        if athlete_name == "Anonymous Athlete":
            continue
        row_data = [athlete_name.split(", ")[0],
        athlete_name.split(", ")[1], athlete_id] + other_data +
        [date_text_clean, city]

        # SELECTION DE VARIABLES uniquement avec un temps qui
        donne plus de 400 pts FINA
        if int(row_data[7]) < 400:
            return all_results
        all_results.append(row_data)
        # Increment the start value for the next page
        start += increment
        time.sleep(random.uniform(10, 20))
    return all_results

def retrieve_and_insert_swimmers(conn, url, start, end, increment,
gender):
    # Retrieve all swimmers from the given URL
    results = retrieve_swimmers(url, start, end, increment)

```

```

for item in results:
    swimmer = {
        's_id': item[2],
        'lastname': item[0],
        'firstname': item[1],
        'birthyear': item[3],
        'sex': gender, # Gender-specific based on the call
        'nationality': item[4],
        'last_club': item[5],
        'pb': item[6]
    }

    insert_swimmer(conn, swimmer)

def extract_splits(onmouseover_attr):
    """
    Extracts split times from the onmouseover attribute.
    Returns a list of tuples (distance, time) or None if no splits
    found.
    """
    onmouseover_attr.split("()")[1].split("')")[0]
    table = BeautifulSoup(onmouseover_attr, "html.parser")

    splits = table.find_all('tr')

    all_splits = []
    last_dist = "0m"

    for split in splits:
        split = [td.text for td in split.find_all('td')]
        if split[0] == '':
            continue
        subdist = f'{int(split[0][:-1]) - int(last_dist[:-1])}m'

        split_dist = split[0]
        split_time = split[1]
        subsplit_dist = subdist
        subsplit_time = split[2]

        all_splits.append({"split_dist": split_dist, "split_time": split_time,
                           "subsplit_dist": subsplit_dist,
                           "subsplit_time": subsplit_time})
        last_dist = split[0]
    return all_splits

def retrieve_race(conn, s_id):
    # Some races are less relevant (see ChatGPT discussion 'Analyse
    # résultats natation')

```

```

    # styles = {2: ["100m", "Freestyle"], 5: ["400m", "Freestyle"],
18: ["200m", "Medley"], }
    # styles = {1: ["50m", "Freestyle"], 2: ["100m", "Freestyle"],
3: ["200m", "Freestyle"], 5: ["400m", "Freestyle"], 6: ["800m",
"Freestyle"], 8: ["1500m", "Freestyle"],
    #         9: ["50m", "Backstroke"], 10: ["100m",
"Backstroke"], 11: ["200m", "Backstroke"],
    #         12: ["50m", "Breaststroke"], 13: ["100m",
"Breaststroke"], 14: ["200m", "Breaststroke"],
    #         15: ["50m", "Butterfly"], 16: ["100m", "Butterfly"],
17: ["200m", "Butterfly"],
    #         20: ["100m", "Medley"], 18: ["200m", "Medley"], 19:
["400m", "Medley"]}

    styles = {2: ["100m", "Freestyle"], 3: ["200m", "Freestyle"], 5:
["400m", "Freestyle"],
        10: ["100m", "Backstroke"],
        13: ["100m", "Breaststroke"],
        16: ["100m", "Butterfly"],
        18: ["200m", "Medley"]}

for style in styles.keys():
    url =
"https://www.swimrankings.net/index.php?page=athleteDetail&athleteId
="+s_id+"&styleId="+str(style)
    distance = styles[style][0]
    stroke = styles[style][1]

    ua = UserAgent()
    headers = {'User-Agent': ua.random}

    start_time = time.time()
    response = requests.get(url, headers)

    if time.time() - start_time > 10:
        print("Took to long to find results for swimmer :
"+s_id+" for style : "+ str(style))
        break
    print("Swimmer : "+s_id+" for style : "+ str(style))
    soup = BeautifulSoup(response.text, 'html.parser')

    # 2 tables : 25m and 50m pool
    table = soup.find("table", class_="twoColumns")
    if table == None:
        print(table)
        print("No results for swimmer : "+s_id+" for style : "+
str(style))
        continue
    for t in table.find_all("table", class_="athleteRanking"):

        pool = "25m" if t.find("th").text.find('25') > 0 else
"50m"

```

```

        results = t.find_all('tr', class_=['athleteRanking0',
'athleteRanking1'])

        # Each table : all results
        for result in results:
            cells = result.find_all('td')

            other_data = [cell.text.strip() for cell in cells]
            r_time = other_data[0]
            r_points = other_data[1]
            r_date = other_data[2].replace('\xa0', ' ')
            r_place = other_data[3]
            row_data = {"s_id": s_id, "distance": distance,
"stroke": stroke, "pool": pool, "time": r_time, "points": r_points,
"race_date": r_date, "place": r_place}

            # Handle split-times
            time_link = result.find('a', class_='time')
            onmouseover_attr = time_link.get('onmouseover')
            splits = extract_splits(onmouseover_attr)

            r_id = insert_race(conn, row_data)

            for split in splits:
                insert_split(conn, r_id, split)

        time.sleep(random.uniform(10, 20))
        update_swimmer_results(conn, s_id)
        print("Swimmer "+str(s_id)+" done")

def update_swimmer_results(conn, s_id):
    update_swimmer_has_results(conn, s_id)

def retrieve_and_insert_races(conn):
    # swimmers = get_todo_swimmers(conn)
    swimmers = get_f_todo_swimmers(conn)
    for swimmer in swimmers:
        retrieve_race(conn, swimmer[0])

# Base URL where the results are listed
m_url =
"https://www.swimrankings.net/index.php?page=rankingDetail&rankingCl
ubId=9268305&firstPlace="
f_url =
"https://www.swimrankings.net/index.php?page=rankingDetail&rankingCl
ubId=9268848&firstPlace="

# Initialize the starting point (1, 26, 51,...)
start = 1
m_end = 12000

```

```

f_end = 12635
increment = 25

if __name__ == "__main__":
    conn = connect_to_db()

    if conn:
        # Create the tables only if needed
        create_tables(conn)

        # get all Belgian Women (based on 100m freestyle, since it
        # is the most known competition)
        # retrieve_and_insert_swimmers(conn, m_url, start, m_end,
        increment, "M") # ALL MEN retrieved

        # retrieve_and_insert_swimmers(conn, f_url, start, f_end,
        increment, "F") # ALL WOMEN retrieved

    retrieve_and_insert_races(conn)

    print("get races")
    races = get_races(conn)
    # db_from_club()
    conn.close()

```

c. Dockerfile.db, image Docker SQL

```

# Use the official PostgreSQL image from the Docker Hub
FROM postgres:latest

# Expose the PostgreSQL port
EXPOSE 5432

```

d. Dockerfile.py, image Docker Python

```

# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the current directory contents into the container at
#/usr/src/app
COPY . .

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Run s.py when the container launches
CMD ["python", "s.py"]

```