|  |
| --- |
| Clermont Foot 63 |
| Manuel du développeur |
| Application Python développée par Nathan Talbot – Stage été 2024 |

|  |
| --- |
| Nathan Talbot |

# ***Principe de l’application***

Site web

Application

Python (Streamlit)

Traitement

des données

Importation des données

Table des matières

[***Principe de l’application*** 1](#_Toc179575141)

[1) IMPORTATION DES DONNÉES 3](#_Toc179575142)

[2) TRANSFORMATION DES DONNÉES 5](#_Toc179575143)

[3) L’APPLICATION STREAMLIT 7](#_Toc179575144)

[4) FAIRE FONCTIONNER L’APPLICATION 8](#_Toc179575145)

[5) AJOUTER DES NOUVELLES SAISONS AUX DONNÉES 12](#_Toc179575146)

# IMPORTATION DES DONNÉES

Dans un premier temps, il faut importer les données Statsbomb et Skill Corner qui seront utilisées pour faire fonctionner l’application.

Les programmes d’importation se trouvent dans le dossier « Données brutes ».

Les données importées proviennent uniquement des saisons de Ligue 2.

1. ***Les codes Python***

Pour chaque programme d’importation, j’ai créé une liste (ou un dictionnaire en fonction du besoin) contenant les saisons disponibles en fonction des différentes APIs.

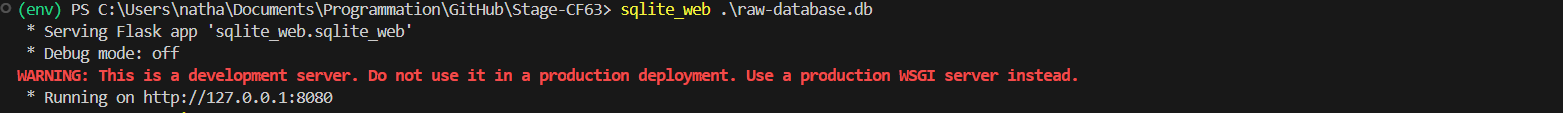
Ensuite, je réalise une boucle traversant l’ensemble des saisons et j’importe les données pour chaque saison dans un dataframe et je concatène l’ensemble de ces dataframes.

Ces données sont ensuite stockées dans une base de données brutes nommée « raw-database », qui n’est pas présente dans le repository Github (je l’ai mise dans le fichier « .gitignore ») car elle est trop volumineuse et n’est pas nécessaire au fonctionnement de l’application.

Pour l’importation via l’API competition\_event de Stats Bomb, je ne garde qu’une partie des données importées avant de les stockées dans la BDD en filtrant les colonnes nécessaires. Cela permet de gagner de l’espace et du temps d’écriture dans la BDD.

De plus, pour cette API, je remplace les colonnes contenant des localisations (liste de coordonnées [x, y, z]) par 3 colonnes contenant respectivement les coordonnées x, y et z. Cela permet de manipuler facilement ces coordonnées, ce qui n’est pas possible lorsque les 3 coordonnées sont stockées dans une même liste.

La manipulation de la BDD est possible grâce à la bibliothèque SQLite3 qui permet d’effectuer des requêtes SQL en Python. Pour visualiser une BDD dans son navigateur, il faut utiliser la bibliothèque « sqlite-web » et taper la commande « sqlite\_web *nom-database » :*



Pour chaque programme d’importation, je créé des variables qui correspondent à mes identifiants Skill Corner/Stats Bomb, ces dernières étant nécessaire pour importer les données de ces fournisseurs. Les mots de passes Statsbomb et Skill Corner sont stockées dans des variables d’environnement sur mon PC.

1. ***Les APIs***

Il faut requêter plusieurs APIs de Skill Corner et de Statsbomb en fonction des besoins des différents modules de l’application :

1/ Métriques discriminantes

2/Évolution des métriques par journée

3/Évolution des métriques par saison

* SK.get\_matches
* SK.get\_match
* SK.get\_in\_possession\_passes
* SK.get\_physical
* SK.get\_in\_possession\_on\_ball\_pressures
* SK.get\_in\_possession\_off\_ball\_runs
* SB.team\_season\_stats

Les données get\_matches et get\_match sont importées via le programme « match\_round.py » se trouvant dans le sous-dossier « Skill Corner ». Elles sont utilisées dans le but d’obtenir le numéro de journée associé à chaque match disponible dans les données Skill Corner. Ces numéros de journée sont nécessaires au fonctionnement du module « Évolution des métriques par journée ».

Je n’importe pas les numéros de journée pour les données des métriques Stats Bomb car les données étudiées sont basées sur une saison et non sur un match, il n’est donc pas possible de différencier les différentes journées.

4/Nombre de passe avant un but

5/Heatmap des zones de début d'action avant un but

6/Heatmap des zones de tir

7/Heatmap des zones de centres

* SB.matches
* SB.competition\_events

L’API SB.matches permet d’obtenir des informations sur les matchs tel que la date, le résultat, la saison etc.

Notes : En ce qui concerne les « physical data » dans mon cas, ces données sont stockées dans des tableaux Excel présents dans les sous dossiers « 2021\_2022 », « 2022\_2023 » et « 2023\_ 2024 » du dossier « Métriques Team sur une Saison Ligue 2 SB + SK ». En effet, à un moment donné, je n’avais plus accès à l’API « get\_physical », et je ne pouvais donc plus importer ces données via le programme « physical.py ». Or, j’avais au préalable stockées ces données dans des tableaux Excels, ce qui explique la présence de ces derniers dans le projet.

Dans le cas où vous avez accès à ces données, vous pouvez supprimer les Excel du projet et utiliser le programme d’importation « physical.py ».

# TRANSFORMATION DES DONNÉES

La transformation des données permet de traité les données importées étant inutilisables, et de ce fait obtenir des données qui pourront être utilisées par l’application Streamlit.

Les programmes sont stockés dans le dossier « Transformation data ».

Ces données brutes sont donc extraites de la BDD « raw-database », puis sont stockées dans la BDD « database » une fois traitées. Cette dernière est beaucoup plus légère que la BDD brutes, et peut donc être stockée dans le repository Github.

Voici comment sont reliés les différents modules de l’application avec les programmes de transformation des données :

* Zone de tir.py
* Passes avant un but.py

4/Nombre de passe avant un but

5/Heatmap des zones de début d'action avant un but

6/Heatmap des zones de tir

* Début action.py

7/Heatmap des zones de centres

* Centre.py
* passes.py
* physical.py
* pressure.py
* running.py
* metriques.py (pour les métriques Stats Bomb

1/ Métriques discriminantes

2/Évolution des métriques par journée

3/Évolution des métriques par saison

Pour les programmes « passes.py », « physical.py », « running.py » ainsi que « pressure.py », j’ai créé une liste « match\_drop » correspondant aux match\_id Skill Corner des matchs de barrages, de play-offs etc. ne faisant pas partis de la saison régulière. En effet, ces matchs posent des problèmes vis-à-vis du module « Évolution des métriques par journée ».

Notes : Dans le cas des « Physical data », étant donné que les données sont stockées dans des Excel, le principe de transformation des données est légèrement modifié. En effet, les données ne sont pas stockées dans la « raw-database » et j’ai donc modifié le code de traitement des données du programme « physical.py ». J’ai laissé en commentaire le code permettant de traiter les données dans le cas ou elles seraient stockées dans la « raw-database ».

En dessous de ce code commenté, j’ai rajouté un code adapté aux Excel et permettant de traiter les « Physical data » de la même façon que si elles seraient stockées dans la database.

Dans ce dernier cas, il faut commenter le code adapté aux Excel et décommenter celui adapté à la « raw-database ».

# L’APPLICATION STREAMLIT

La partie développement de l’application (front-end) se trouve dans le dossier « Application\_streamlit ».

Le dossier «.streamlit » se trouvant à la racine du repository contient un fichier config.toml permettant de configurer des caractéristiques de base de l’application tel que la couleur de fond (je l’ai fixé en « light » pour l’application).

Dans le dossier « Application\_streamlit », on y retrouve :

* Le fichier « Main.py » : il sert de fichier d’entrée à l’application ; c’est lui qui est exécuté au lancement de cette dernière. Dans mon cas, je m’en sers pour initialiser les différents « session states » qui seront utilisés par les multiples modules (documentation sur les « sessions states » : <https://docs.streamlit.io/develop/api-reference/caching-and-state/st.session_state>), mais aussi pour définir l’ensemble des pages disponibles (1 page = 1 module).
* Le fichier « variable.py » contenant les variables étant partagées entre les pages. Ce fichier contient en particulier les dictionnaires « dico\_rank\_SK » et « dico\_rank\_SB » qui contiennent chacun les classements des équipes pour chaque saison disponible en fonction du fournisseur de données. Il y’a un dictionnaire par fournisseur car les noms d’équipes Skill Corner et Stats Bomb sont différents.
* Le fichier « fonction.py » contient l’ensemble des fonctions python qui sont utilisées par les différentes pages.
* Le dossier « apps » contient l’ensemble des pages/modules de l’application.

Grand principe des programmes Streamlit :

* A chaque fois que l’utilisateur interagit avec un widget, le programme est relancé de haut en bas.
* Lorsque l’utilisateur actualise son navigateur ou change de page, les valeurs des widgets sont réinitialisées.
* Les sessions states permettent de stocker les valeurs des widgets en mémoire et donc de garder les choix effectués pour ceux-ci lors d’un changement de page ou actualisation du navigateur.
* La commande « @st.cache\_data » permet d’enregistrer le résultat d’une fonction en fonction des paramètres qui lui sont passés : cela permet de garder en mémoire le résultat d’une fonction dans le cas où on viendrait à l’appeler une deuxième fois en utilisant les mêmes paramètres que pour le premier appel, et donc de ne pas à avoir à réexécuter cette fonction (pratique dans le cas où une fonction met longtemps à s’exécuter, comme par exemple une importation de données).

# FAIRE FONCTIONNER L’APPLICATION

Dans un premier temps, il faut importer le repository Github en local sur son ordinateur via Github Desktop.

Note : l’ensemble des manipulations que je vais expliquer sont réalisées grâce à l’éditeur VS code, je conseille donc fortement de l’utiliser pour faciliter la mise en place de l’application.

1. ***Environnement virtuel***

Il est nécessaire créer l’environnement virtuel qui va servir d’interpréteur à l’ensemble des programmes. L’environnement n’est pas présent de base dans le repository du fait de son volume : je l’ai placé dans le fichier « .gitignore ».

Pour créer ce dernier via VS code, il faut dans un premier temps ouvrir un terminal de commande et accéder à la racine du repository.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Ensuite, suivre les étapes suivantes en entrant ces commandes dans le terminal :

* Créer l’environnement : python -m venv env
* Lancer l’environnement : .\env\Scripts\activate
* Installer les bibliothèques Python nécessaires au fonctionnement de l’ensemble des programmes : pip install -r requirements.txt

Attention : si la ligne « pywin32==307 » est présente dans le fichier « requirements », il faut la supprimer car elle pose un problème lors de la publication de l’application.

* Pour chaque programme Python et Jupyter Notebook, il faut définir l’environnement virtuel « env » comment interpréteur de ces derniers. Pour les programmes Python, il faut ouvrir la « command palette » (ctrl + shift + p) :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Ensuite, chercher « Python :Select Interpreter » et choisir « Python 3.12.6 (‘env’ : venv) »

Pour les programmes Jupyter, il suffit de cliquer sur « Detecting Kernels » (ou l’interpréteur actuel du programme) et sélectionner « env (Python 3.12.6) » :

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Description générée automatiquement

Important : l’environnement virtuel doit nécessairement se situer à la racine du projet.

1. ***Exécution des programmes d’importation (Facultative)***

Cette étape est facultative : la base de données traitées est déjà présente dans le repository et suffit au fonctionnement de l’application. Mais dans le cas où l’utilisateur à besoin d’importer de nouvelles données, de changer de compétition ou autre, voici comment faire fonctionner ces programmes :

* Dans un premier temps, penser à modifier les identifiants Stats Bomb et Skill Corner pour l’ensemble des programmes, et définir les variables d’environnements « mdp\_skillcorner » et « mdp\_statsbomb » sur son pc.
* Lancer l’ensemble des programmes d’importation.

Ces étapes doivent normalement créer à la racine du projet la « raw-database » (et uniquement cette database, sinon il doit très certainement y avoir un problème quant aux interpréteurs de ces programmes).

1. ***Exécution des programmes de transformation (Facultative)***

Dans le cas ou de nouvelles données ont été importées, il est donc nécessaire de transformer ces dernières. Pour ce faire, il suffit de :

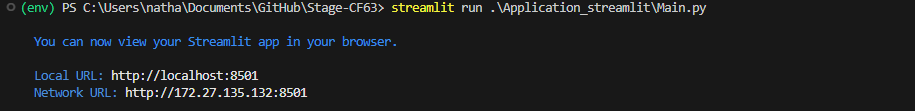
* (Par précaution) Supprimer l’actuelle « database.db ».
* Lancer l’ensemble des programmes présents dans le dossier « Transformation data ». Ces programmes permettent de créer la nouvelle « database.db », qui doit se trouver à la racine du projet.

Une fois les étapes précédentes effectuées, l’application est prête à être utilisée. Il est possible entre autres de la lancer localement, ou de la publier sur le site de Streamlit.

1. ***Lancer l’application en local***

Il est possible de lancer l’application en local, elle sera alors hébergée sur une adresse locale.

Pour ce faire, il faut entrer la commande « streamlit run .\Application\_streamlit\Main.py », en ayant pensé à activé l’environnement virtuel au préalable, et il faut se trouver à la racine du projet.



L’application réagira alors à chaque modification et sauvegarde d’un des modules/programmes.

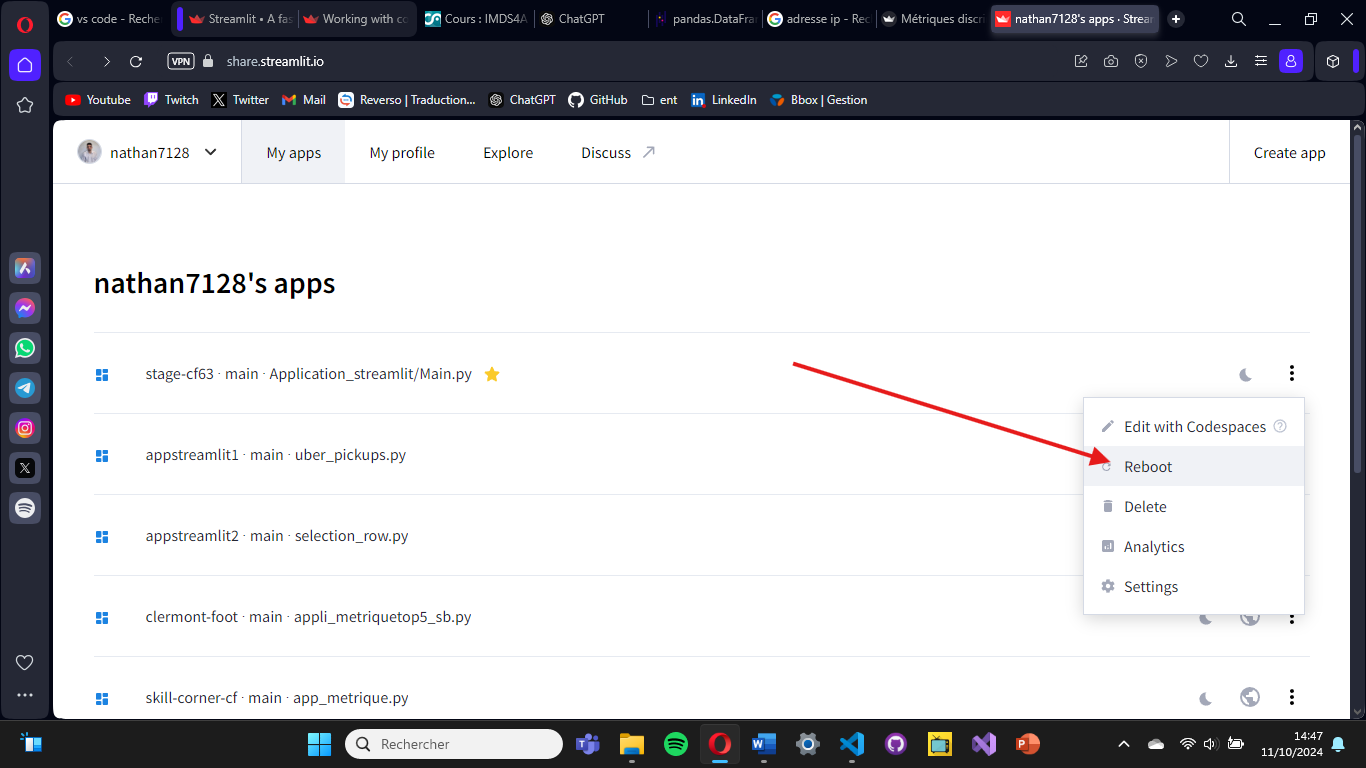
Cette méthode est utile pour effectuer des modifications sur l’app.

1. ***Héberger l’application via le site de Streamlit***

Lorsque l’application est prête à être publiée, il est alors possible de publier l’application via le site de Streamlit. Le principe est le suivant : l’application est liée à un repository Github ; il est donc essentiel de créer un repository semblable au mien et de le publier sur son compte Github.

Pour ce faire, il faut accéder au site de Streamlit (<https://streamlit.io>), se connecter ou créer un compte lié à son compte Github, cliquer sur « Create app » en haut à droite, cliquer sur « Yup, I have an app », sélectionner le bon repository dans l’emplacement « Repository », sélectionner « Application\_streamlit/Main.py » dans « Main file path », et éventuellement choisir un url personnalisé pour cette app.

A chaque modification et « push » de cette dernière sur le repository, Streamlit détectera cette dernière et l’application sera mise à jour.

Attention : Il arrive parfois que des changements ne soient pas détecter, il faut alors reboot l’app : 

Pour régler la confidentialité de l’application en privé, il faut choisir l’option « Only specific people can view this app », entrer les mails des personnes qui auront accès à l’app dans la section « Invite viewers by email » et cliquer sur « Save ».

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

# AJOUTER DES NOUVELLES SAISONS AUX DONNÉES

Il est possible d’ajouter des saisons supplémentaires lors de l’importation des données. Cependant, pour ce faire, il faut effectuer quelques modifications aux codes du projet.

1. ***Importation des données***

Une seule modification est nécessaire : il faut ajouter pour l’ensemble des programmes d’importations la saison (ou le couple saison/saison\_id) aux listes de saisons (ou au dictionnaire).

1. ***Traitement des données***

La seule modification à effectuer est d’ajouter aux programmes « passes.py », « physical.py », « running.py » et « pressure.py » les match\_id Skill Corner des matchs de barrages pour les saisons ajoutées.

1. ***Application Streamlit***

Au niveau des programmes de l’application Streamlit, il est nécessaire d’ajouter les classements des équipes (pour les saisons ajoutées) aux dictionnaires « dico\_ranking\_SK » et « dico\_rank\_SB » du programme « variable.py ».

Cette méthode d’ajout de saison nécessite donc de relancer l’ensemble des programmes d’importation et de transformation de données pour mettre à jour la « database.db » contenant les données utilisées par l’application à chaque mise à jour de ces dernières. Ce qui n’est donc pas optimal dans le cas où l’on voudrait ajouter seulement un match à la base de données.