SAE2.04- Exploitation d'une base de données

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL

iut
CRÉTEIL-VITRY

DÉPARTEMENT INFORMATIQUE

1. Introduction

Ce rapport présente le projet réalisé dans le cadre de la SAE2.04 - Exploitation d'une base de donnée. L'objectif de ce projet était de développer une application permettant d'accéder aux données historiques du système de Vélos en Libre-Service (VLS) de la métropole parisienne. Cette application offre la possibilité d'observer et d'analyser ces données à différentes échelles spatiales et temporelles, ouvrant ainsi la voie à une meilleure compréhension et à l'amélioration des services de la ville. Dans ce rapport, nous décrirons tout d'abord le contexte du projet, puis nous aborderons les différentes étapes de l'acquisition et du stockage des données, la mise en place de la partie web, ainsi que l'analyse des données collectée. Enfin, nous conclurons en présentant les perspectives d'amélioration pour ce projet.

2. Contexte

Cette application vise à fournir un accès aux données historique du système de Vélos en libre-service (VLS) de la métropole parisienne, permettant ainsi d'observer ces données à différentes échelles spatiales et temporelles. L'objectif est de mieux comprendre, analyser et finalement améliorer les services de la ville en utilisant une vision historique des données. Grâce à l'ouverture des données en temps réel pour les vélos en libre-service parisiens, il est possible de constituer une base de données historiques contenant les données enregistrées et diffusées par ce système.

Cette application constitue une première étape avant l'analyse des données brutes collectées dans ce contexte. Elle offre différentes visualisations permettant de comparer les différentes observations pour une commune spécifique ou une station donnée.

3. Acquisition et stockage des données

3.1. Source et emplacement des données

Il s'agit de la première étape. Les données sont disponibles sans clé et au format JSON en UTF-8 en accord avec la RFC 4627. Elles sont mises à jour chaque minute selon la norme GBFS1.0. Elles sont à disposition sur le site open data Paris. Le mouvement Open Data vise à rendre accessibles et utilisables les données électroniques détenues par les organisations, y compris les collectivités publiques. Dans cet esprit, la Ville de Paris a décidé d'ouvrir ses données publiques, permettant ainsi aux chercheurs, développeurs, citoyens, journalistes et entreprises de les exploiter pour des travaux, des services innovants, des informations brutes et des créations à valeur ajoutée. Depuis janvier 2011, de nombreux jeux de données sont disponibles sur le site Open Data Paris, et cette démarche s'inscrit dans une politique d'innovation ouverte et de co-conception avec les habitants.

C'est dans la continuité de cette démarche que s'inscrit cette SAÉ. Les données sont divisées en deux ensembles : les données statiques qui restent constantes, telles que l'emplacement de la station, son nom, son identifiant, etc., et les données dynamiques qui varient, telles que le nombre de vélos disponibles ou l'état de disponibilité des stations, c'est-à-dire si elles sont ouvertes ou fermées.

3.2. Structure et stockage des informations

Afin d'analyser et d'améliorer le système de Vélos en libre-service de la métropole parisienne, il est essentiel de disposer d'un historique des données. Pour chaque requête à l'API, il est nécessaire de stocker les résultats quelque part. Le réseau Vélib' étant le plus grand système de VLS au monde, avec ses 1464 stations, chaque requête implique de stocker 1464 lignes de données. J'ai choisi de réaliser des **requêtes toutes les 5 minutes** afin d'obtenir des données précises. Avec 1440 minutes dans une journée, cela représente 288 requêtes par jour. Ainsi, une seule journée génère un volume de données de 421 632 lignes (288 x 1464), soit près d'un million de lignes sur une période de deux jours.

Pour gérer efficacement ce volume de données, l'utilisation d'une base de données est primordiale. C'est pourquoi cette SAÉ utilise le système de gestion de base de données (SGBD) MySQL. MySQL est réputé pour ses performances élevées, ce qui en fait le choix optimal pour cette application.

MySQL est un serveur de base de données relationnelles SQL, ce qui implique le besoin de structurer les données sous forme de table et de relation. Dans le cadre de ce projet, j'ai créé deux tables pour les données et une table pour consigner toutes les actions qui se déroulent dans la base de données (log). Je vais ici parler des deux tables principales, nous reviendrons sur la table de log ultérieurement dans ce rapport.

La base de données est donc constituée d'une table nommé « station_information » qui est la représentation en SQL du jeu de donnée statique, elle contient l'identifiant de la station, son nom, sa capacité, ses coordonnées et un petit ajustement personnel, sa commune. En effet, dans le jeu de données d'origine, la commune est classée comme une donnée dynamique, ce qui aurait entraîné un volume de données considérable. Par conséquent, j'ai pris la décision de l'inclure dans la table « station_information » en tant qu'ajustement personnel, afin d'éviter une multiplication excessive des données.

Ensuite, la deuxième table, nommée « station_status », concerne les données qui peuvent changer régulièrement. Elle est composée d'un champ « date » qui enregistre la date et l'heure de la requête effectuée sur la base de données. Elle comprend également un identifiant qui est une clé étrangère reliant les données statiques concernant cette station, en cas de suppression des données statiques d'une station, toutes les données associées sont aussi supprimées. Les autres champs de la table « station_status » incluent l'état d'ouverture ou de fermeture de la station, le nombre de places de stationnement disponibles, le nombre total de vélos disponibles, le nombre de vélos mécaniques disponibles et le nombre de vélos électriques disponibles. Cette table permet de stocker les informations dynamiques qui peuvent varier fréquemment.

Le schéma relationnel représentant ces tables est disponible en annexe (Figure 1).

3.3. Déclencheurs et intégrité des données

L'intégrité des données est une préoccupation majeure lors de l'utilisation d'une base de données. Pour garantir cette intégrité, j'ai mis en place plusieurs déclencheurs (triggers) spécifiquement conçus à cet effet.
Tout d'abord, il existe deux déclencheurs d'intégrité référentielle sur la colonne « stationcode » de la table dynamique. Ces déclencheurs sont essentiels car en cas de suppression d'une station de la base de données, la clé étrangère correspondante ne serait plus associée à aucune ligne de la table statique. Cela pourrait entraîner des problèmes lors des requêtes de sélection. Afin de résoudre ce problème, les déclencheurs sont générés par MySQL en utilisant les paramètres « ON DELETE CASCADE » et « ON UPDATE CASCADE ». Ainsi, lorsque la référence à une station est supprimée ou mise à jour, toutes les correspondances dans la table dynamique sont également supprimées ou mises à jour de manière cohérente.
De plus, un autre déclencheur est mis en place pour vérifier que la colonne « is_installed » de la table dynamique ne peut prendre que les valeurs « OUI » ou « NON ». Aucune autre valeur n'est autorisée. Cette vérification permet de garantir que seules ces deux états sont présents dans la base de données lors des opérations de sélection.
Ces déclencheurs contribuent à maintenir l'intégrité des données en veillant à ce que les relations référentielles soient respectées et en limitant les valeurs possibles pour certaines colonnes. Ainsi, la base de données reste cohérente et les requêtes peuvent être effectuées en toute confiance.

3.4. Création et gestion de la BDD en Python

Maintenant que la partie théorique sur la gestion des données est réglée. En pratique, comment ça marche ? Le fonctionnement pratique du script repose sur l'utilisation d'une classe appelée « DbConnector ». Cette classe est responsable de la gestion de la connexion à la base de données, de la vérification des tables et des permissions, ainsi que de toutes les opérations nécessaires sur la base de données (voir annexe, Figure 2 pour un exemple de sortie lors de l'initialisation de la classe « DbConnector »).

La classe « DbConnector » est conçue en utilisant le design pattern Singleton, ce qui signifie qu'elle ne peut être instanciée qu'une seule fois dans tout le script, peu importe le nombre de fois où son constructeur est appelé. À chaque appel du constructeur, il renvoie simplement la même instance de la classe (tout en vérifiant les permissions fournies dans le constructeur à chaque fois). Cela est possible grâce à l'utilisation d'une méta classe pour « DbConnector » ; une méta classe est une classe en python qui hérite de « type » qui est appelé lors de l'instanciation d'une autre classe. En somme, elle permet de définir des comportements particuliers lors de l'instanciation d'une classe (voir Figure 3).

Cela permet de garantir une seule connexion à la base de données tout au long du script, évitant ainsi les connexions multiples et améliorant l'efficacité de l'accès aux données.

Grâce à la classe « DbConnector », les opérations sur la base de données, telles que l'exécution de requêtes SQL, l'insertion de données, la mise à jour des enregistrements, etc., peuvent être effectuées de manière pratique et centralisée.

Le fichier principal du script se base sur une bibliothèque appelée « Click », qui permet de créer facilement un CLI pour une application python (voir Figure 4). Il prend en argument un fichier d'environnement dans lequel on retrouve les informations d'authentification à la BDD ; à la suite du fichier d'environnement, il y'a le choix entre plusieurs commandes possibles certaines d'entre elle prennent des arguments aussi.

À chaque exécution du script, la classe « DbConnector » vérifie si les tables nécessaires existent dans la base de données. Si une table est manquante et que les permissions appropriées sont détectées, la classe « DbConnector » procède à sa création. Ainsi, toute commande exécutée avec le script a la capacité de créer les tables dans une base de données vide.

La création des tables s'appuie sur une autre classe appelée « Tables », qui hérite de la classe « Enum » de Python. La structure de chaque table est définie de manière similaire à un objet JSON, permettant de spécifier les noms des colonnes, les types de données et les contraintes associées à chaque table. Pour faciliter ce processus, une fonction est utilisée pour convertir cette structure de données en une requête SQL appropriée. C'est aussi à cette étape que les triggers sont créés de la même manière.

Lors de la première exécution du script, la première commande à exécuter est « initdb ». Cette commande récupère les données statiques à partir de la source de données ouverte (open data) et les insère dans la table « station_information ». Ainsi, les informations telles que l'identifiant de la station, son nom, sa capacité, ses coordonnées et sa commune sont enregistrées dans cette table.

Une fois les données statiques insérées, la commande suivante à exécuter est « adddata ». Cette commande récupère les données dynamiques, qui peuvent varier régulièrement, et les insère dans la table « station_status ».

De plus, deux autres commandes sont disponibles bien qu'utilisée rarement ; il s'agit de « resetdb » et « resetlog », ces deux commandes sont assez explicites, la première permet de supprimer toutes les données et de recréer les tables, la seconde permet de supprimer de recréer la table de log.

3.5. Déploiement du script

Maintenant que le script python est prêt, il faut faire en sorte de le lancer toutes les 5 minutes et ce sur plusieurs jours. J'ai la chance de posséder des serveurs dans des data centers ce qui m'a grandement facilité l'opération.

La première étape consiste à compiler le script Python en une archive Wheel. Pour ce faire, il faut créer un module Python en ajoutant un fichier « __init__.py » vide. Ensuite, il est nécessaire de créer un fichier « pyproject.toml » contenant toutes les informations relatives au projet, telles que les dépendances, le fichier principal à exécuter, la version de Python utilisée, etc.

Une fois ces étapes terminées, il suffit d'exécuter une commande Python pour obtenir une archive Wheel. L'avantage de cette approche réside dans le fait que sur un serveur, il suffit d'utiliser « pip » en spécifiant le chemin de l'archive pour pouvoir exécuter le script avec une simple commande. Ainsi, le déploiement est rapide et efficace.

Une fois l'installation finies, il me suffit d'utiliser la crontab sur mon serveur pour lancer le script d'ajout des données dynamique toutes les 5 minutes (voir Figure 5). De plus, le script met également à jour les données statiques une fois par heure. A chaque exécution, la sortie de la commande est ajoutée à un fichier de log (voir Figure 6 et Figure 27).

4. Partie Web

4.1. Technologie utilisée

Maintenant que les données sont récupérées et stocké dans la BDD. Il faut un moyen de les consulter. Elles seront disponibles via un navigateur web. Pour la partie web, j'ai utilisé Next.js qui est un framework de développement web basé sur React qui offre plusieurs avantages significatifs pour la création d'applications web.

Tout d'abord, Next.js facilite la création d'une API REST en utilisant les fonctions dites « API Routes ». Ces routes peuvent être définies dans notre application Next.js pour gérer les requêtes HTTP et accéder aux données de notre base de données. Il est possible de créer des endpoints personnalisés qui renvoient les données sous forme de JSON, ce qui permet à notre navigateur web d'interagir avec l'API et de récupérer les informations nécessaires.

De plus, Next.js permet de gérer à la fois le côté client et le côté serveur au même endroit. Cela signifie qu'il est possible d'utiliser les mêmes composants React pour construire à la fois l'interface utilisateur et les fonctionnalités côté client. Cela simplifie le processus de développement en évitant la nécessité de créer une application client séparée et de gérer la synchronisation des données entre le client et le serveur.

Next.js offre également des fonctionnalités telles que le rendu côté serveur (SSR) et le rendu côté client (CSR) hybride. Cela permet d'optimiser les performances de notre application en générant les pages côté serveur pour le contenu initial, puis en passant au rendu côté client pour les interactions dynamiques. Cela permet une expérience utilisateur fluide tout en bénéficiant des avantages du rendu côté serveur pour améliorer le référencement et l'accessibilité de notre application.

4.2. Pages disponibles

L'application web se compose de plusieurs pages offrant différentes fonctionnalités. La page d'accueil propose un graphique qui affiche le nombre de stations par commune, ainsi qu'une liste des stations+ (plus d'informations) faisant partie du réseau Vélib'. Cette page d'accueil est assez rudimentaire, fournissant une vue globale des informations (voir Figure 7).

La page principale de l'application présente une carte interactive, accompagnée de filtres de recherche permettant aux utilisateurs de rechercher des stations spécifiques par nom ou par commune. Sur cette page, un bouton est disponible pour basculer entre les données en temps réel et les données moyennes sur une période choisie par l'utilisateur (voir Figure 8 et Figure 9).

Sur la carte, quel que soit le mode sélectionné, les informations suivantes sont affichées pour chaque station : nom, identifiant, capacité et commune. Dans le cas du mode de données moyennes, la carte affiche également le nombre moyen de vélos disponibles (électriques et mécaniques) ainsi que le nombre moyen de places disponibles. En revanche, pour le mode de données en temps réel, la carte affiche le nombre actuel de vélos disponibles (électriques et mécaniques), ainsi que le nombre de places disponibles en temps réel.

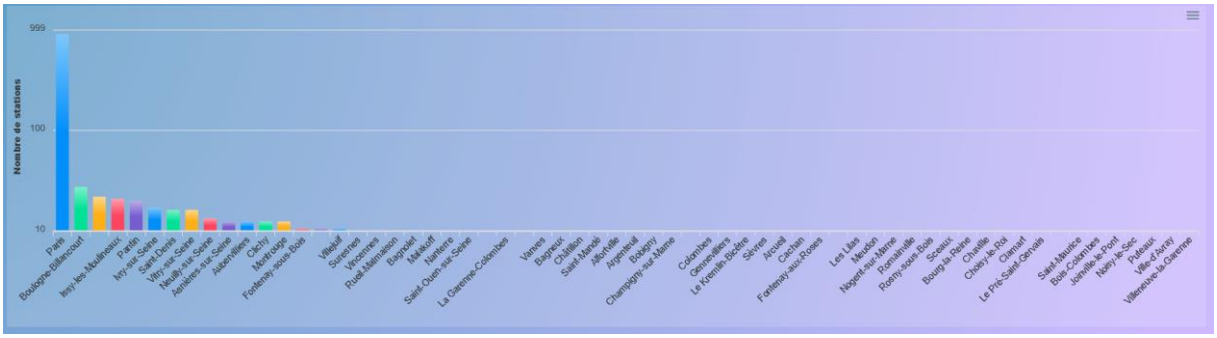
De plus, lorsqu'un utilisateur clique sur une station spécifique, il peut accéder à des statistiques détaillées ainsi qu'à l'historique des données relatives à cette station (voir Figure 10).

De plus, plusieurs graphiques ont été ajoutés en complément des cartes. Pour les données en temps réel, les stations les plus remplies et les stations les moins remplies sont affichées en dessous de la carte. Sous la carte des données moyennes, on retrouve le flux total de vélos pour l'ensemble du réseau ou la commune sélectionnée, ainsi qu'un camembert illustrant la répartition des vélos électriques et des vélos mécaniques. De plus, un histogramme représentant cette répartition est également présenté pour chaque jour. Le nombre total de stations pour le réseau ou la commune sélectionnée est également indiqué. Enfin, une section intitulée « déplacements pertinents » est présente et sera exploré ultérieurement dans ce rapport.

5. Analyse

5.1. Rôle de Paris dans le réseau Vélib’

Lorsque nous examinons le premier graphique, il est évident que la grande majorité des stations Vélib' se trouve à Paris. Sur les 1464 stations Vélib' au total, 999 sont situées à Paris, ce qui représente environ 68% de toutes les stations.



Cette concentration du réseau Vélib' autour de Paris se reflète également dans le trafic total des stations. Pour mesurer le trafic total d'une station, j'utilise une méthode qui consiste à calculer la somme des valeurs absolues des différences entre le nombre de vélos disponibles à l'instant T et le nombre de vélos disponibles à l'instant T-1.

En d'autres termes, pour chaque station, je compare le nombre de vélos disponibles à deux moments consécutifs et j'enregistre la différence absolue entre ces deux valeurs. Ensuite, je fais la somme de toutes ces différences absolues pour obtenir le trafic total de la station. Cette mesure nous donne une indication du niveau d'activité et de mouvement qui se produit au sein de chaque station.

Lorsque nous analysons le trafic total des stations Vélib' en le comparant par commune, nous constatons que Paris occupe une place prépondérante. En effet, en calculant le trafic total par commune, nous observons que Paris représente environ 78% de l'ensemble du trafic (voir Figure 11 et Figure 12).

Cette donnée met en évidence l'importance capitale de Paris dans le réseau Vélib'. La concentration des stations dans la capitale entraîne naturellement un niveau de trafic plus élevé par rapport aux autres communes. Cela s'explique par la densité de population, l'activité économique et touristique de la ville, ainsi que par l'efficacité et la popularité du système de vélos en libre-service dans cette zone.

Cependant, malgré le rôle central de Paris dans le réseau Vélib', il est intéressant de constater que lorsqu'on compare le taux de remplissage moyen et la taille des stations par commune, Paris ne se distingue pas nécessairement dans les résultats. Il convient néanmoins de prendre en compte le fait que Paris possède quelques-unes des plus grandes stations, mais cela reste des exceptions.

Cela signifie que si l'on considère la capacité moyenne des stations pour chaque commune, Paris ne se démarque pas particulièrement. En effet, la capacité moyenne des stations pour l'ensemble des communes se situe autour de 31, avec un écart-type de 11. Cela indique que la capacité des stations dans les autres communes est généralement similaire à celle de Paris (comme on peut le voir ici Figure 14), avec quelques variations. En résumé, Paris se distingue par le nombre de ses stations plutôt que par leur taille.

Pour ce qui est du taux de remplissage, les données sont un peu plus dispersées avec une moyenne à 36% et un écart-type de 19%. Paris est un peu en dessous de la moyenne mais ne se distingue pas forcément (voir Figure 13)

5.2. Part des vélos électrique

En 2018, un nouveau type de vélo a été introduit dans le service de vélos en libre-service Vélib'. Il s'agit des Vélib' électriques, qui ont apporté une nouvelle dimension aux déplacements en offrant une assistance électrique aux utilisateurs. Ces vélos électriques ont rapidement gagné en popularité, offrant la possibilité de couvrir de plus longues distances et de surmonter les reliefs urbains sans effort supplémentaire. Cette évolution dans le service Vélib' a permis de rendre les déplacements plus simples et accessibles à un plus grand nombre d'usagers. Dans cette analyse, nous nous intéresserons à la part des vélos mécaniques et électriques dans les mouvements globaux du réseau Vélib'.

En effectuant la moyenne des nombres totaux de vélos électriques et de vélos mécaniques disponibles dans les stations Vélib', nous pouvons calculer la proportion de vélos électriques par rapport à l'ensemble des vélos du réseau. Sur la dernière période mensuelle, cette proportion s'élève à 37%, ce qui est assez proche des chiffres officiels communiqués par Vélib' indiquant une proportion de 40% de vélos électriques dans leur flotte (voir Figure 17 et site officiel de Vélib'). Ainsi, ces résultats confirment que les vélos électriques occupent une part significative dans le réseau Vélib', offrant aux utilisateurs la possibilité de bénéficier d'une assistance électrique lors de leurs déplacements.

La question que je me suis posée est la suivante : « Dans quelle mesure les utilisateurs préfèrent-ils les vélos électriques par rapport aux vélos classiques ? » Pour y répondre, j'ai analysé les mouvements au sein des stations en les distinguant entre les mouvements impliquant des vélos électriques et ceux impliquant des vélos classiques. Les résultats sont clairs : même si seulement 37% des vélos sont électriques, ils représentent près de 56% des mouvements (voir Figure 15 et Figure 16). Cela démontre de manière évidente que dans la plupart des cas, lorsqu'un utilisateur a le choix

entre un vélo classique et un vélo électrique, il aura tendance à opter pour le vélo électrique. Cette préférence s'explique probablement par les avantages offerts par les vélos électriques, tels que la facilité des déplacements sur de longues distances ou en terrain vallonné, sans nécessiter d'efforts physiques importants.

Une option d'amélioration potentielle pour le réseau Vélib' serait d'augmenter la part des vélos électriques. Étant donné que les utilisateurs semblent préférer les vélos électriques aux vélos classiques, une augmentation de l'offre de vélos électriques pourrait susciter davantage d'intérêt et encourager une utilisation plus fréquente du service. Cela pourrait se traduire par une meilleure satisfaction des utilisateurs, une augmentation du nombre de trajets effectués et une réduction potentielle de la congestion urbaine.

La question suivante que j'ai examinée était de savoir si Vélib' mettait en place une stratégie visant à augmenter la part des vélos électriques dans son réseau. Pour répondre à cette question, j'ai analysé l'évolution de la part des vélos électriques sur une période de plus d'un mois. Malheureusement, mes observations n'ont révélé aucun changement significatif dans la proportion de vélos électriques dans le réseau Vélib', ni même dans le nombre total de vélos disponibles (voir Figure 18 et Figure 19). Cela suggère que, pour le moment, Vélib' n'a peut-être pas mis en œuvre de stratégie spécifique pour augmenter la part des vélos électriques. Cependant, il convient de noter que ces résultats ont été obtenus sur une période relativement courte, ce qui pourrait masquer une tendance qui se poursuivraient sur des échelles temporelles plus longue.

5.3. Flux de vélos et heure de pointe

En examinant le réseau Vélib', nous constatons qu'il est composé d'environ 19 000 vélos. Pour parvenir à ce chiffre, j'ai observé le nombre maximal de vélos disponibles sur l'ensemble de la période de mes données, en supposant que le nombre de vélos utilisés est minimal pendant la nuit. Cependant, il est important de noter que cette estimation peut être légèrement sous-estimée, car il pourrait y avoir des usagers utilisant Vélib' à 3 heures du matin. Malgré cette approximation, j'ai confiance en mon chiffre étant donné qu'il est raccord avec les chiffres communiqués par Vélib' (voir ici)

En moyenne, si nous calculons la différence entre le moment où il y a le plus grand nombre de vélos disponibles et le moment où il y en a le moins chaque jour, nous obtenons un chiffre de 4 227. Cela signifie que même pendant les heures de pointe, 77 % des vélos restent inutilisés.

Cette constatation soulève la question de l'efficacité de l'utilisation des vélos disponibles dans le réseau Vélib'. Bien que de nombreux vélos soient disponibles, une grande partie d'entre eux reste inutilisée pendant une grande partie de la journée. Cependant, il convient de noter que même dans le cas d'un vélo personnel, il serait rarement utilisé pendant 23% de la journée. Ainsi, il est compréhensible que certains vélos restent inutilisés à certains moments de la journée.

Il est important de souligner que, dans l'ensemble, le réseau Vélib' demeure bien plus pratique et efficace que si chaque usager devait posséder son propre vélo. Grâce à Vélib', les utilisateurs peuvent facilement accéder à un vélo lorsque nécessaire, sans se soucier de son entretien, de son stockage ou des éventuelles réparations. De plus, cela encourage l'utilisation des transports actifs et contribue à réduire la congestion routière et les émissions de carbone. Malgré les vélos inutilisés à certains moments, le réseau Vélib' reste une solution pratique et durable pour la mobilité urbaine.

La question suivante que j'ai abordée concerne les heures de pointe dans le réseau Vélib'. Comme la plupart des autres moyens de transport, on pourrait présumer que Vélib' connaît une heure de pointe le matin, mais les résultats m'ont surpris. En examinant le nombre total de vélos disponibles dans le

réseau à chaque instant de mes données, et en ne considérant que les heures où ce nombre est inférieur à la moyenne moins l'écart-type, j'ai identifié les moments où le réseau Vélib' était très sollicité. En additionnant les heures de ces moments, j'ai constaté que l'heure de pointe s'étend de 15h à 18h, avec un pic à 16h, tandis que 7h du matin ne se classe qu'en cinquième position (voir Figure 20). Cela suggère que les utilisateurs de Vélib' sont davantage motivés à faire du vélo dans l'après-midi plutôt que le matin.

5.4. Analyse par stations

Pour approfondir mon analyse, je me suis penché sur les données de stations Vélib' au niveau des stations individuelles. Tout d'abord, j'ai examiné de près une station Vélib' près de chez moi, appelée « Rouget de Lisle », que je connais assez bien. En observant son historique de données, j'ai remarqué que son activité connaît des pics qui pourraient correspondre aux heures d'arrivée des RER (voir Figure 21).

Ensuite, j'ai envisagé qu'une station Vélib' située à proximité d'un établissement ouvert la nuit pourrait connaître une activité plus soutenue pendant cette période. C'est pourquoi j'ai analysé l'historique de la station « Clichy - Place Blanche », qui se trouve à proximité du Moulin Rouge. Effectivement, j'ai constaté que l'activité de cette station se poursuit tout au long de la nuit (voir Figure 22). Cette observation met en évidence l'un des aspects clés du réseau Vélib' par rapport à d'autres moyens de transport : sa disponibilité 24h/24.

De plus, j'ai entrepris d'analyser une station située en altitude, car il est évident que les utilisateurs de Vélib' n'ont pas besoin de remonter les vélos en haut des pentes. Il est donc logique de supposer que les vélos des stations en altitude seront davantage utilisés pour descendre que pour monter. En examinant l'historique des données de la station « Belleville – Pyrénées » située dans les hauteurs de Paris. Cette tendance se confirme, la station est généralement vide, étant principalement approvisionnée par l'équipe Vélib' (caractérisé par des changements rapides du nombre de vélos disponible) ou par des utilisateurs de vélos électriques courageux (voir Figure 23). En outre, personne n'aurait envie de faire l'ascension à vélo classique. En effet, une analyse de la répartition des vélos dans cette station révèle que 88% des vélos disponibles sont électriques, ce qui est nettement supérieur à la moyenne de 37% dans l'ensemble du réseau (voir Figure 24).

Il convient de souligner que l'équipe Vélib' a pris des mesures pour résoudre ce problème spécifique lié aux stations en altitude. En effet, afin d'encourager les utilisateurs à remonter les vélos dans ces stations, un système de minutes bonus a été mis en place. Ainsi, en ramenant un vélo dans une station située en hauteur, les utilisateurs bénéficient de minutes supplémentaires à utiliser pour prolonger leur utilisation d'un Vélib'. Cette incitation intelligente encourage les usagers à effectuer l'effort de remonter les vélos, tout en optimisant l'utilisation des vélos électriques disponibles dans ces stations (voir ici). Cependant, cela n’a pas l’air d’avoir totalement réglé le problème, et je n’ai malheureusement pas les données d’avant le 10 janvier 2023 pour comparer l’efficacité de cette mesure.

5.5. Tentative d’analyse des déplacements de vélos pertinents

Le bon fonctionnement du service Vélib' repose sur une équipe dédiée chargée de maintenir un équilibre au niveau des stations, évitant ainsi qu'elles se retrouvent vides ou surchargées. Leur mission consiste à redistribuer les vélos entre les stations en fonction des fluctuations de la demande. Par exemple, lorsque certaines stations tendent à se remplir, des vélos sont déplacés depuis des stations moins fréquentées, telles que celles en altitude. Le rôle de cette équipe est essentiel, et l'analyse des déplacements nécessaires pourrait même constituer un projet à part entière. La méthodologie que j'ai utilisée consiste à évaluer le taux de remplissage de chaque station à une

heure spécifique, dans ce cas précis à 22h. Ensuite, j'ai calculé la moyenne de ces taux pour chaque station. Si la moyenne était inférieure à 10%, j'ai identifié la station la plus proche ayant un taux de remplissage moyen supérieur à 80%. Cette approche, bien qu'elle ne soit pas suffisamment sophistiquée pour être utilisée dans la pratique réelle, a permis de fournir quelques idées intéressantes concernant les déplacements pertinents des vélos entre les stations (voir Figure 25).

6. Conclusion

En conclusion, ce projet SAE2.04 a été une expérience enrichissante et instructive. Grâce au développement de cette application permettant l'accès aux données historiques du système de Vélos en Libre-Service de la métropole parisienne, j'ai pu acquérir de nouvelles compétences en matière d'acquisition et de stockage des données, ainsi qu'en développement web et en analyse de données. Cette analyse des données m'a permis d'observer des tendances intéressantes et d'en apprendre davantage sur le fonctionnement du réseau Vélib'. Bien que ce projet ait atteint ses objectifs, il est important de souligner que l'analyse de ces données pourrait se poursuivre indéfiniment, ouvrant la voie à de nouvelles perspectives et opportunités d'amélioration des services de la ville. Je suis satisfait des résultats obtenus jusqu'à présent et je suis convaincu que cette application constitue une base solide pour la compréhension et l'analyse du réseau Vélib'

7. Perspective d'améliorations

Bien que cette analyse des données du réseau Vélib' ait apporté des résultats satisfaisants, il existe certaines perspectives d'amélioration à considérer. Tout d'abord, l'étude aurait pu être approfondie en prenant en compte le profil social des usagers, ce qui aurait permis de mieux comprendre leurs besoins et attentes spécifiques. Malheureusement, les données disponibles ne comprenaient pas ces informations, limitant ainsi notre analyse. Il serait intéressant d'explorer davantage cette dimension sociale dans de futurs projets, en utilisant des enquêtes ou d'autres sources de données complémentaires. De plus, étant donné que c'était ma première expérience avec un framework basé sur React pour le développement de l'application web, je reconnais que le code du projet pourrait bénéficier d'optimisations supplémentaires. Néanmoins, cette expérience m'a permis d'acquérir des compétences précieuses en développement React et d'approfondir ma compréhension de cette technologie. Je suis confiant dans ma capacité à améliorer et à optimiser davantage le code à l'avenir.

Cependant, en ce qui concerne mon code python, j'en suis très satisfait. Je pense même que la classe « DbConnector » pourrait me resservir un jour étant donné sa modularité.

8. Digression sur la fonction NOW() et la duedate

Au début du projet, j'ai été confronté à la question de savoir quelle date utiliser, celle de récupération des données ou celle présente dans la colonne « duedate » des données dynamiques. Au départ, j'ai pensé que la colonne « duedate » serait plus précise. Cependant, j'ai remarqué un comportement étrange avec cette colonne qui était mise à jour une fois par heure. Cela m'a amené à remettre en question si les données étaient réellement mises à jour toutes les minutes, comme annoncé. J'ai donc créé deux bases de données : l'une utilisant la colonne « duedate » des données dynamiques, et l'autre utilisant la fonction NOW() de SQL (le script prend un paramètre pour décider quelle méthode utiliser). D'après mes résultats, les données étaient bien mises à jour à intervalle d'une minute. Il s'est avéré que soit je n'avais pas compris le but de la colonne « duedate », soit celle-ci présentait un dysfonctionnement. J'ai donc contacté le Syndicat Autolib' Vélib' par e-mail pour obtenir des éclaircissements. Plus tard, Ils m'ont confirmé qu'il y avait effectivement un problème avec cette

colonne et m'ont suggéré d'utiliser la date de récupération des données à partir de l'API, ce qui s'est avéré être une solution viable (voir Figure 26).

9. Déploiement du site web

Afin de rendre une partie des graphiques utilisés dans ce rapport accessibles, j'ai mis en ligne l'application web correspondante. Pour ce faire, j'ai utilisé la commande de compilation de l'application Next.js, puis j'ai configuré un reverse proxy dans Nginx. Vous pouvez accéder à l'application web à l'URL suivante : https://sae204.nwo.ovh/. Veuillez noter que, en raison de la grande quantité de données et de la complexité des requêtes SQL à exécuter, ainsi que de la puissance relativement limitée de mon serveur Linux, le chargement des différentes vues peut être assez lent, pouvant prendre plus de 3 minutes.

Annexes

← Aparté sur NOW et DUEDATE →

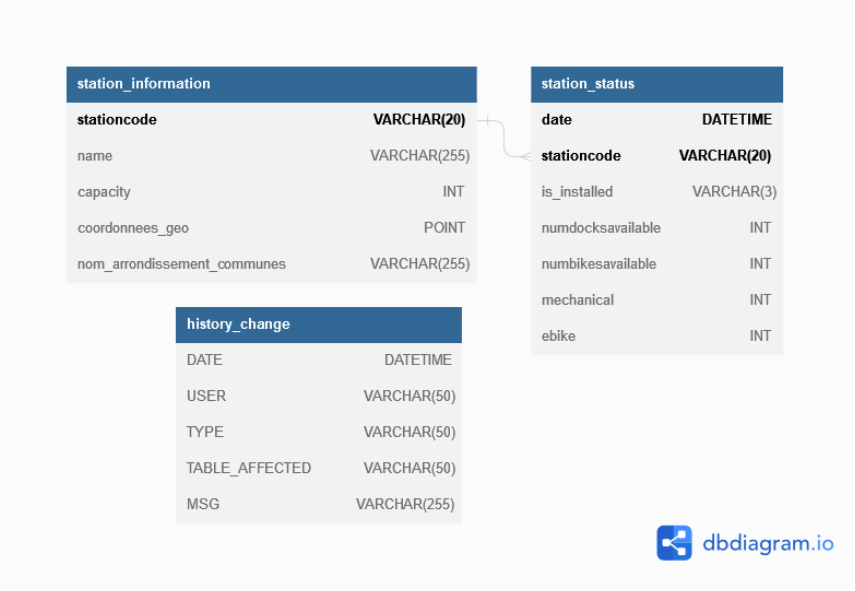


Figure 1 : Schéma relationnel représentant la base de données pour ce projet

```
--- Initialisation de la connexion à la BDD ---
sae204now
Connexion à la base de données sae204now réussie !
--- vérification de la BDD ---
ping...
ping réussi
vérification des permissions...
Récupération des privilèges de l'utilisateur...
vérification des permissions réussie
Vérification des tables...
La table history_change existe
La table station_information existe
La table station_status existe
```

Figure 2 : Script de lancement et de vérification de la BDD

```python
class Singleton(type):
    """
    Metaclass permettant de créer une classe singleton
    Elle s'assure qu'il n'y a qu'une seule instance de la classe

    Une Metaclass est une classe qui définit comment une classe est créée, elle est appelée lorsque la classe est créée
    """
    _instances = {}
    # Lorsque la metaclass est appelée
    def __call__(cls, *args, **kwargs): # cls = Nom de la classe, *args = liste des arguments, **kwargs = liste des arguments nommés
        if cls not in Singleton._instances:
            Singleton._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)

        if hasattr(cls, "verification_method"):
            # appel la méthode de vérification de la classe (dans le cas de DbConnexion, c'est la méthode qui vérifie que les permissions données en argument du constructeur sont bien présentes)
            cls.verification_method(Singleton._instances[cls], *args, **kwargs)
        return Singleton._instances[cls]
```

Figure 3 : Méta classe Singleton

```
(.venv) PS C:\Users\Maxpi\Documents\GitHub\SAE2.04-EXPLOBDD\src> python.exe .\main.py
Usage: main.py [OPTIONS] COMMAND [ARGS]...

  CLI pour la gestion de la base de données à chaque démarrage, vérifie que
  les variables d'environnement sont bien définies et vérifie que les tables
  sont bien créées

Options:
  --env FILE  Environnement à utiliser  [default: .env]
  --debug     Active le mode debug
  --help      Show this message and exit.

Commands:
  adddata   Ajoute des données dynamiques à la base de données
  dropdb    Supprime toutes les tables de la base de données (sans recréer les
            tables)
  initdb    Remplit la base de données avec les données statiques
  initlog   Crée (ou reset) la table de log
  resetdb   Réinitialise la base de données (supprime les données existantes
            et les recrée)
  resetlog  reset la table de log
```

Figure 4 : Aide du CLI

```
*/5 * * * * /home/winteru/SAE204/run_and_log_sae204_now_method.sh
32 * * * * /home/winteru/SAE204/update_static_and_log_sae204_now.sh
```

Figure 5: Crontab

```
[winteru@nwo3 ~]$ cat SAE204/run_and_log_sae204_now_method.sh
#!/bin/bash

# Change the paths to match your script and log file
DIR_PATH="/home/winteru/SAE204"
LOGFILE_PATH="${DIR_PATH}/sae204_now_method.log"

/home/winteru/.local/bin/sae204 --env "${DIR_PATH}/.env2"  adddata --force -d SQLNOW 2>&1 | while IFS= read -r line; do
    printf "[%s] %s\n" "$(date "+%Y-%m-%d %H:%M:%S")" "$line"
done >> "${LOGFILE_PATH}"
```

Figure 6 : Script de lancement

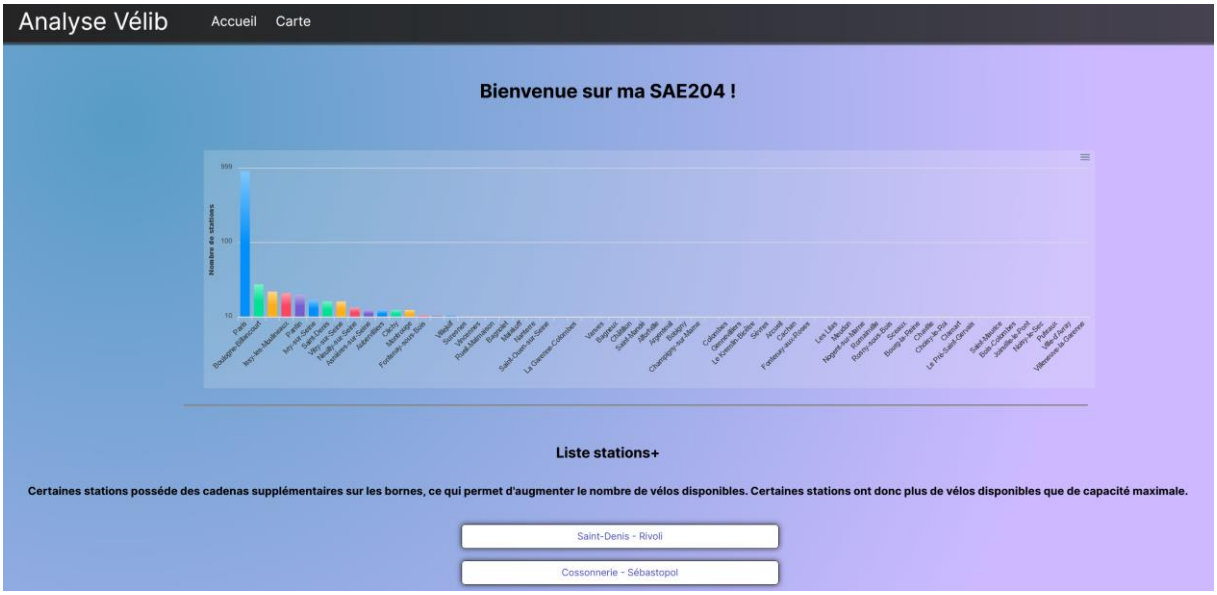


Figure 7 : Application web - Page d'accueil

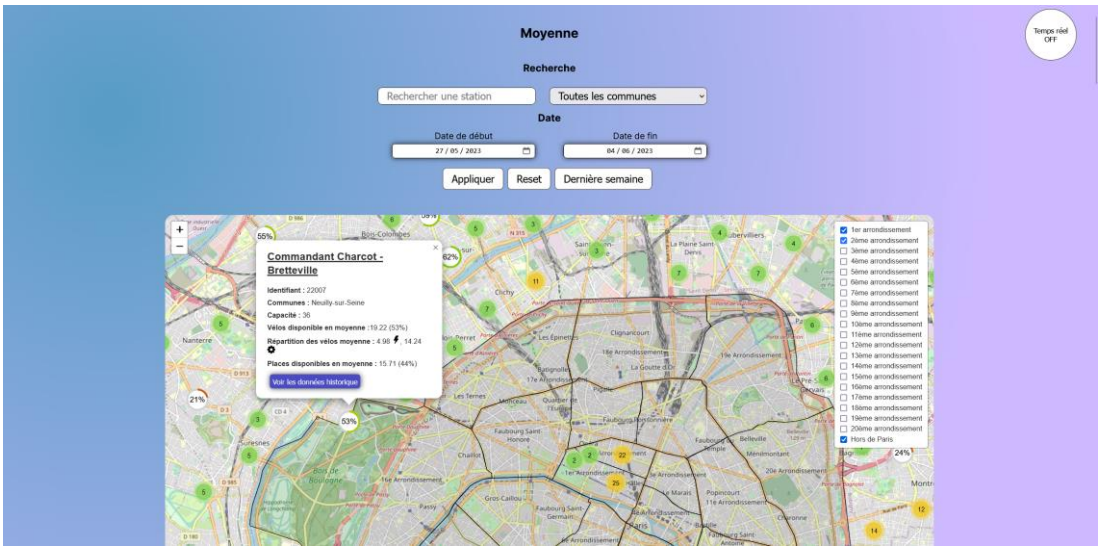


Figure 8 : Carte Moyenne

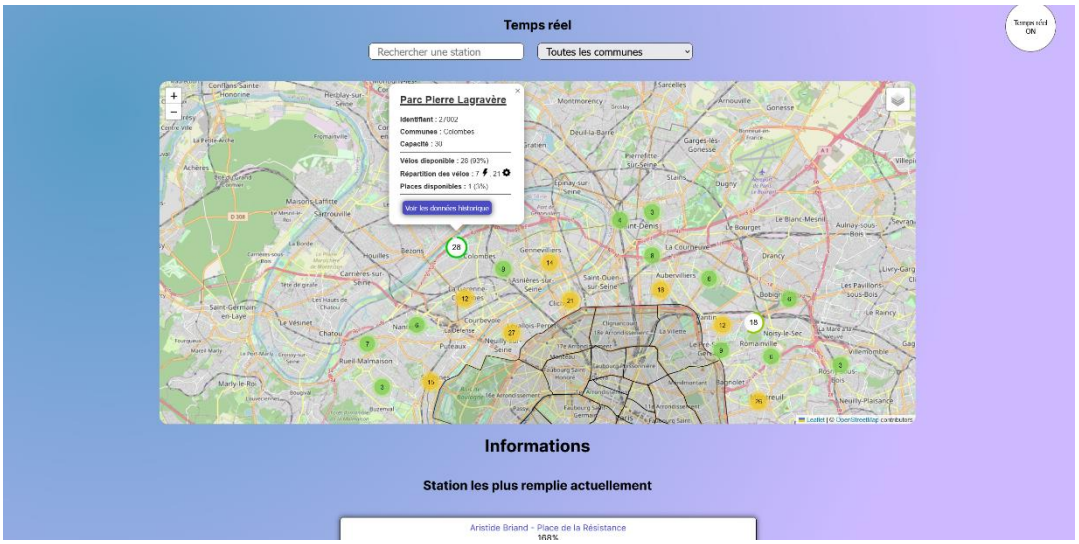


Figure 9 : Carte temps réel

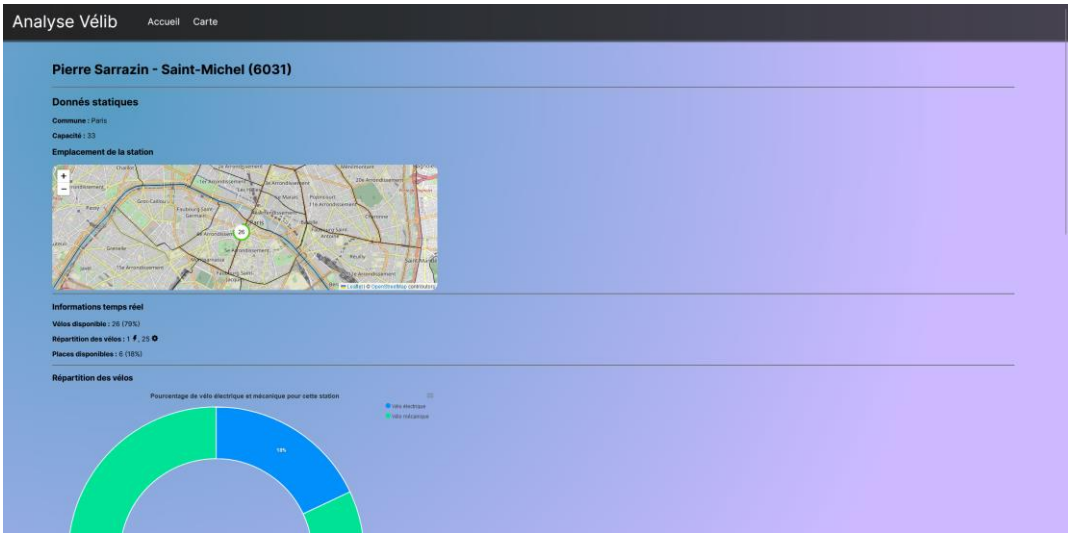


Figure 10 : Affichage par station

```python
# data colonnes : stationcode, total_movement et nom_arrondissement_communes

# regrouper le mouvement total par commune et faire la somme
data2 = data.groupby(['nom_arrondissement_communes'])[["total_movement"]].sum()

# afficher le résultat trié par ordre décroissant
data2.sort_values(by=['total_movement'], ascending=False, inplace=True)

sum_movement = data2['total_movement'].sum()
data2['percentage'] = data2['total_movement'].apply(lambda x: round((x / sum_movement) * 100))
display(data2)
```

✓ 0.0s

nom_arrondissement_communes	total_movement	percentage
Paris	4031114	78
Boulogne-Billancourt	103187	2
Montreuil	75371	1
Pantin	58801	1
Ivry-sur-Seine	58077	1
...	...	...
Argenteuil	2171	0
Choisy-le-Roi	2119	0
Chaville	2072	0
Ville-d'Avray	652	0
Villeneuve-la-Garenne	0	0

Figure 11 : Mouvement total par commune

```sql
SELECT
    stationcode,
    SUM(total_movement) AS total_movement,
    nom_arrondissement_communes
FROM
    (
    SELECT
        si.stationcode,
        ABS(COALESCE(ss.numbikesavailable - LAG(ss.numbikesavailable) OVER (PARTITION BY ss.stationcode ORDER BY ss.date), 0)) AS total_movement,
        si.nom_arrondissement_communes AS nom_arrondissement_communes
    FROM
        station_information AS si
        INNER JOIN station_status AS ss ON si.stationcode = ss.stationcode
    WHERE
        si.capacity > 0
    ) AS subquery
GROUP BY
    stationcode;
```

Figure 12 : Requête SQL permettant de calculer le niveau d'activité pour chaque station

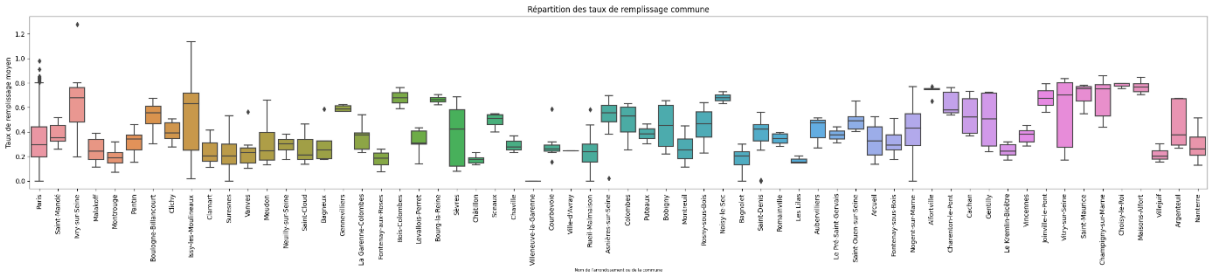


Figure 13 : Répartition du taux de remplissage par commune

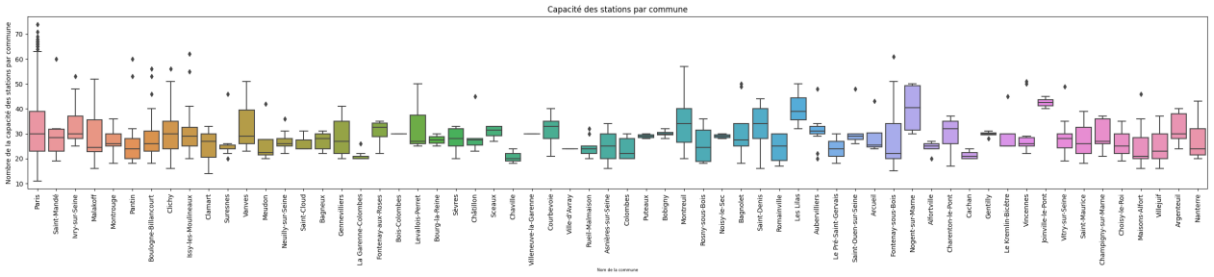


Figure 14 : Capacité des stations par commune

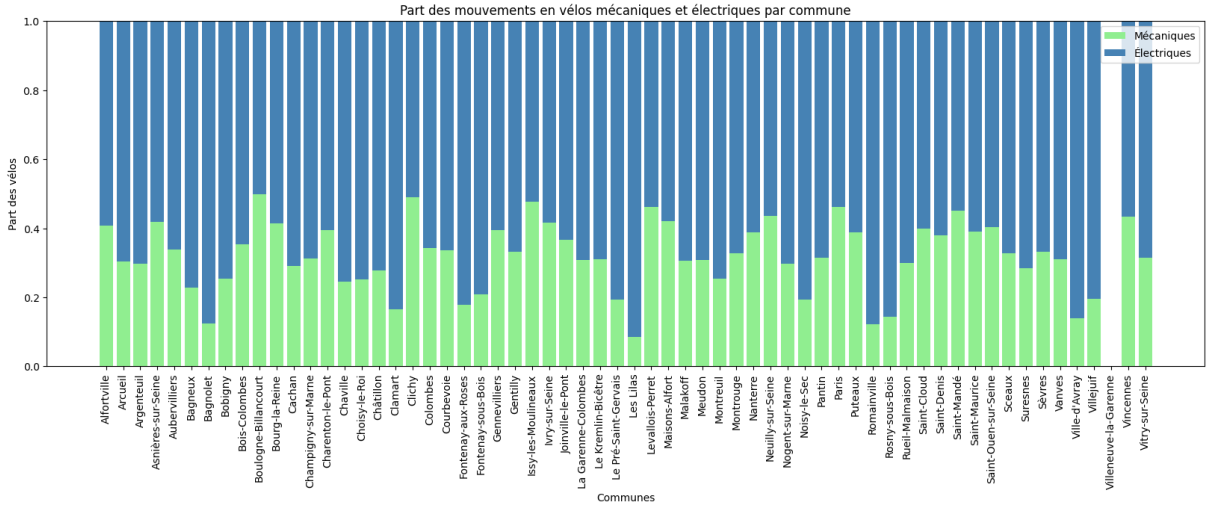


Figure 15 : Part des mouvements des vélos mécaniques et électrique par commune

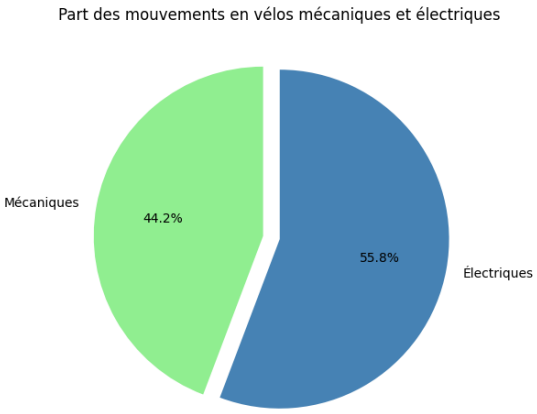


Figure 16 : Part des mouvements en vélos mécaniques et électriques

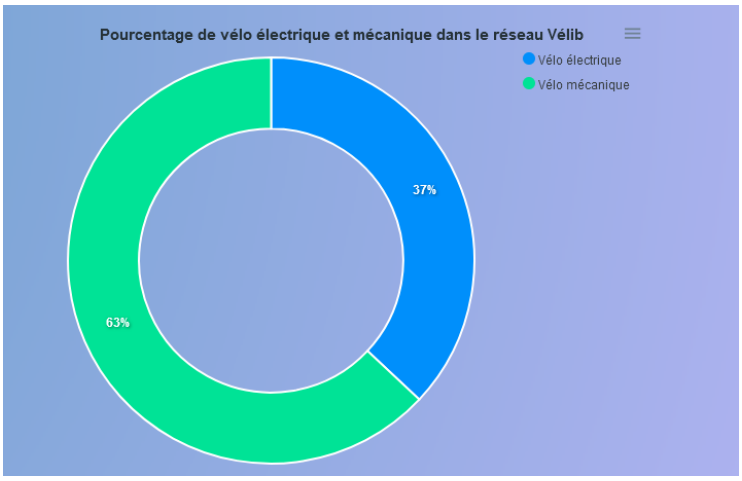


Figure 17 : Part du nombre de vélos électrique et mécanique dans le réseau Vélib'

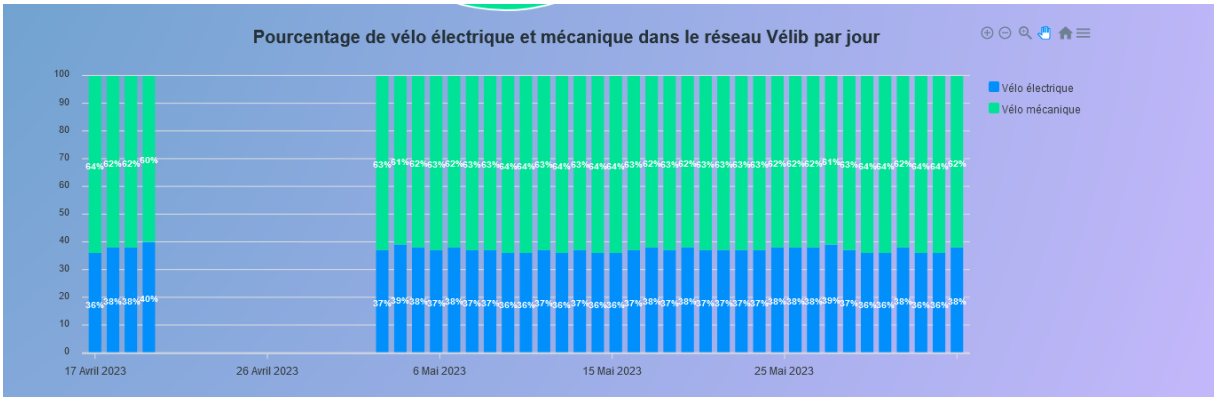


Figure 18 : Part du nombre de vélos électrique et mécanique dans le réseau Vélib' par jour

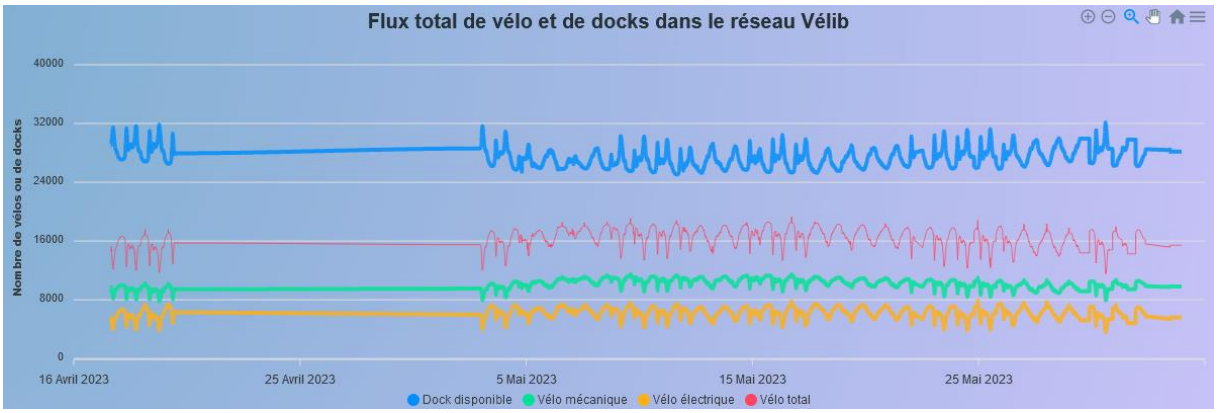


Figure 19 : Flux total de vélo et de docks dans le réseau Vélib'

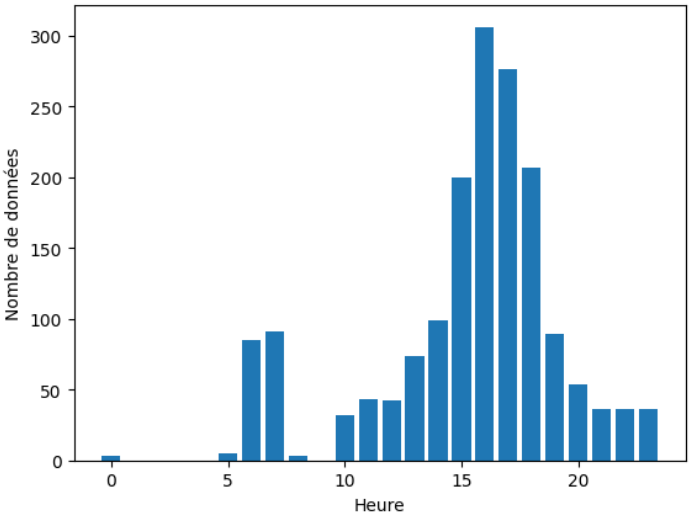


Figure 20 : Nombre de mesure où le nombre de vélos disponibles était inférieur à la médiane - l'écart type groupé par heure

Figure 21 : Historique données Rouget de Lisle (Choisy-le-Roi RER)

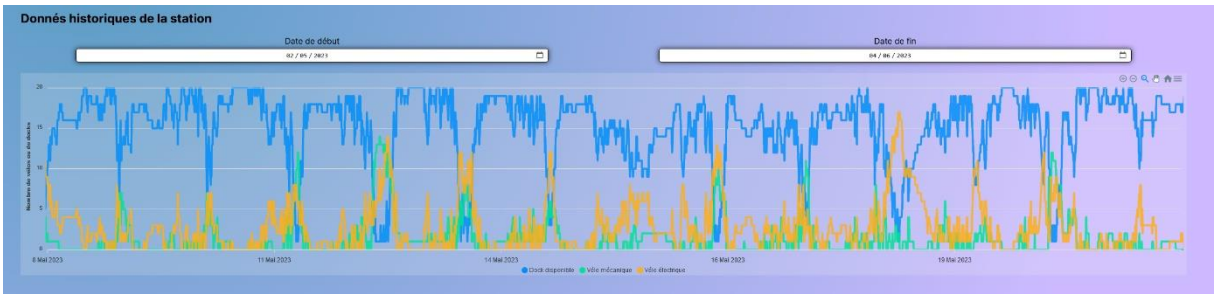


Figure 22 : Historique données Clichy – Place Blanche

Figure 23 : Historique données Belleville – Pyrénées

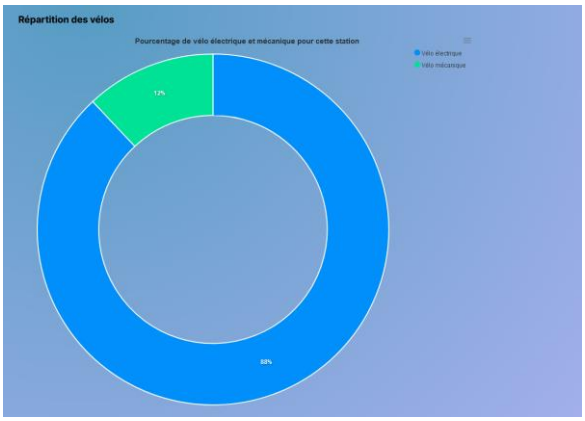


Figure 24 : Répartition des vélos pour la station Belleville – Pyrénées

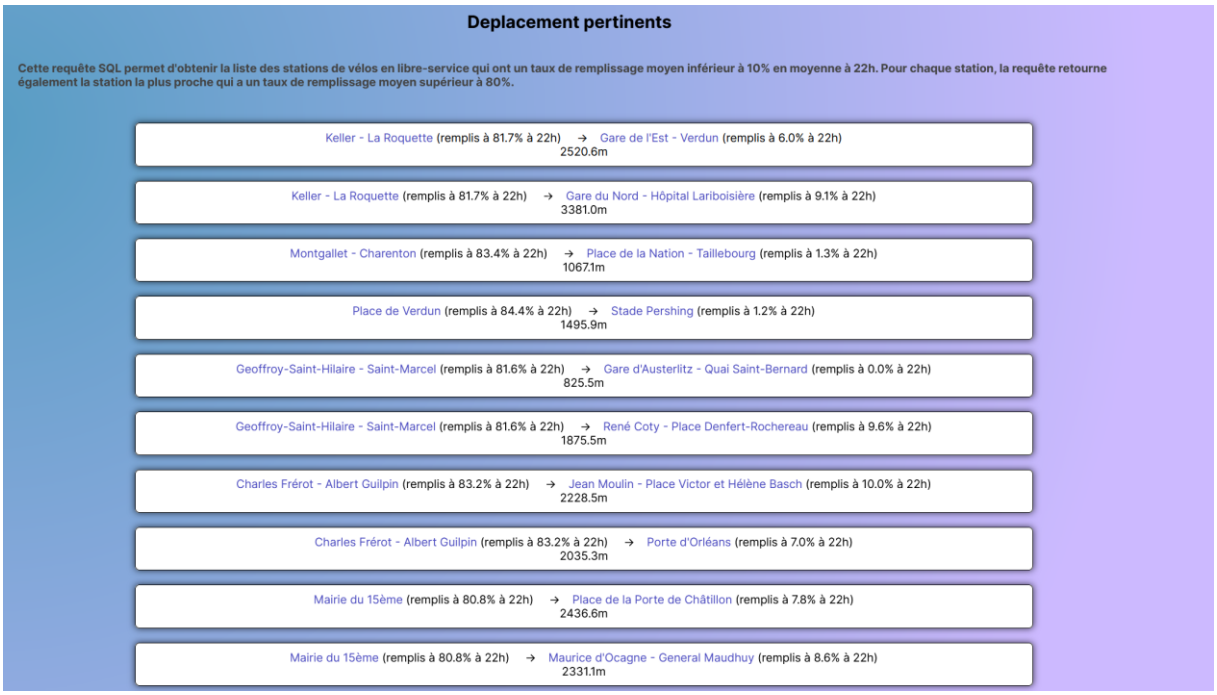


Figure 25 : Déplacement pertinents exemple

Figure 26 : Réponse du Syndicat Autolib' Vélib'

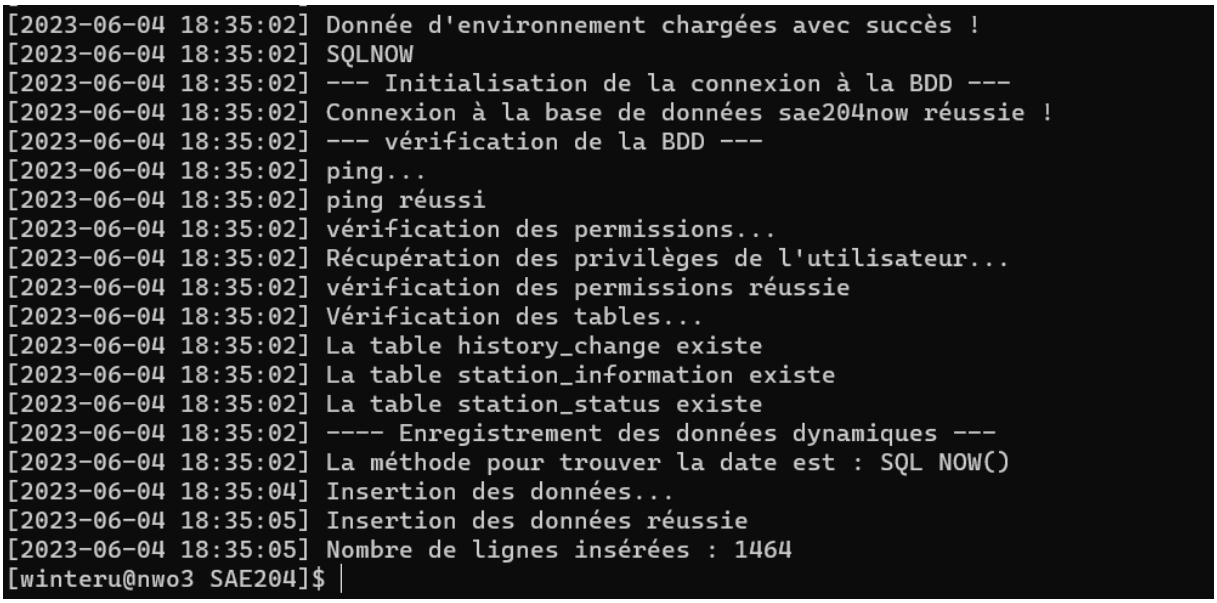


Figure 27 : Exemple de sortie du script lors de l'insertion des données dynamique