Anticipe et optimise tes trajets en VélôToulouse

William SCHNEIDER
Minh-Quân NGUYEN
Guillaume REYNIER
Colin HERVIOU-LABORDE

Motivation de la startup :

Notre vocation est de permettre aux utilisateurs d'anticiper leurs déplacements en VélÔ Toulouse grâce à prédictions de disponibilités et d'emplacements du réseau VélÔ Toulouse.

1) Extraction

1.1) Source des données

Nous avons exploité les données mises à disposition par l'API de JCDecaux, prestataire du service VélôToulouse ainsi que de nombreux autres systèmes de vélos partagés dans le monde. Les données sont fournies au format JSON. À chaque requête, nous récupérons en temps réel les informations relatives à l'ensemble des stations de la ville de Toulouse.
Voici à quoi ressemblent les données après un appel à l'API :


```
{
    "number": 224,
    "contract_name": "toulouse",
    "name": "00224 - BELIN - SUPAERO",
    "address": "10 AV EDOUARD BELIN FACE CHATEAU D'EAU CREPS RD-PT JEAN LAGASSE",
    "position": {
        "lat": 43.56644,
        "lng": 1.475301
    },
    "banking": false,
    "bonus": false,
    "bike_stands": 20,
    "available_bike_stands": 2,
    "available_bikes": 18,
    "status": "OPEN",
    "last_update": 1758718802000
},
```

1.2) Stratégie d'extraction et fréquence de collecte

Mais nous remarquons immédiatement une première limitation : l'API ne permet pas d'accéder à l'historique des données. Or, pour pouvoir effectuer des prédictions, nous avons besoin d'un volume de données étalé sur une période suffisamment longue.
Il va donc être nécessaire de stocker les données de multiples requêtes à l'APU à une fréquence bien choisie.

Une historisation complète et quasi-continue de toutes les stations peut représenter un volume de données qui peut être considérable. Après avoir fait des requêtes toutes les minutes pendant 1 heure nous obtenons un fichier d'une taille de : 760 kO. Nous estimons donc que pour un an, nous aurons un volume de données de 6,7 GO.
Ce volume de données est raisonnable et peut être stocké localement.

Nous avons opté pour une fréquence d'une minute. Ce délai correspond à peu près au temps entre deux actions successives (prise ou dépose) sur une même station. On note que lorsqu'on choisit de mobiliser un vélo, on a une minute pour le sortir de la borne.

1.3) Qualité et fiabilité des données

Les données provenant directement du gestionnaire officiel des stations, il n'existe pas de source plus fiable. Cependant, afin d'assurer leur validité, nous filtrons les doublons de timestamps et de station afin d'écarter d'éventuelles erreurs ou doublons. Nous filtrons également les stations fermées ou inactives, dont les données ne sont plus mises à jour.

1.4) Accès et contraintes techniques

L'accès à l'API nécessite une clé d'authentification, facilement obtenue en créant un compte gratuit sur le site officiel de JCDecaux (https://developer.jcdecaux.com). En dehors de cette clé, aucune contrainte de sécurité particulière n'est imposée et il ne semble pas avoir de limitations en nombre de requêtes pour la fréquence de requête que nous avons définie.

1.5) Perspectives d'enrichissement des données

Dans le but de prédire la disponibilité des bornes et des vélos du réseau, il sera nécessaire de récupérer des données extérieures afin d'entraîner le modèle.

Parmi ces sources potentielles, les données météorologiques occupent une place importante : les conditions météo influencent fortement l'usage des vélos. Nous pourrions ainsi, via une API météorologique.

Un autre axe d'amélioration consisterait à intégrer des informations relatives aux événements se déroulant à Toulouse. Par exemple, la tenue d'un match du Stade Toulousain a un impact direct sur la disponibilité des vélos autour du stade Ernest-Wallon.

2) Transformation

La sortie de la fonction extract est un fichier json, la première étape est de transformer ces données en dataframe *pandas.*

```
# mise au format dataframe pandas
df = pd.json_normalize(data)
```

Il y a un traitement à effectuer sur les échantillons de temps :

- convertir les timestamp bruts en ms en datetime
- mettre ces datetime dans la zone correspondante à l'heure française

```
# correction de l'heure sous le bon format
df['datetime'] = pd.to_datetime(df['last_update'], unit='ms', utc=True)
df['datetime'] = df['datetime'].dt.tz_convert('Etc/GMT-2')
```

Remarque : il faudra adapter le code au changement en heure d'hiver

Ensuite, il suffit de filtrer les stations ouvertes avec la colonne "*status*", puis on ne sélectionne uniquement les colonnes intéressantes pour nous

```
# filtrage des données intéressantes
df = df[df["status"] == "OPEN"]
df = df[["datetime", "number",  "name", "address", "bike_stands", "available_bike_stands", "available_bikes"]]
```

Finalement, on trie le dataframe par échantillon de temps et par numéro de borne croissants afin de préparer au mieux la concaténation des données.

3) Loading

3.1) Choix du système de données

Nous choisissons de charger nos données dans une data warehouse.
Les APIs de JCDecaux ou OpenWeather ne conservent pas forcément l'historique complet.
Une data warehouse permet de :

- stocker les snapshots de disponibilité des vélos dans le temps,
- conserver l'historique météo,
- créer un calendrier des évènements sur Toulouse,
- croiser ces données pour entraîner des modèles prédictifs et générer des analyses BI

et tout cela de façon structurée, c'est-à-dire sous forme de tableau.

3.2) Fréquence de chargement

La fréquence de chargement des données dépend de la nature de ces dernières :

- Les données statiques propres aux stations (numéro de borne, adresse etc…) peuvent être mises à jour dès qu'elles changent. A priori, elles ne devraient pas être beaucoup modifiées au cours du temps
- Les données de disponibilités sont chargées en temps réel, à la minute

- Les données météo toutes les 10 minutes, à l'échelle d'une ville (Toulouse), on peut supposer que la météo est uniforme peu susceptible de changer sur une durée de 10 minutes
- Le calendrier des événements sera mis à jour dans la base de données tous les mois. En effet, les évènements (match de football, rugby, concerts, festivals…) sont souvent planifiés au moins un mois à l'avance

3.3) Vérification des données

On s'assure de la consistance des données et de leur intégrité en vérifiant qu'elles respectent le schéma défini pour la data warehouse. Utiliser des contraintes lors de la définition des relations permet de contrôler le type des données mais aussi d'empêcher la création de doublons non souhaités dans la base de données.

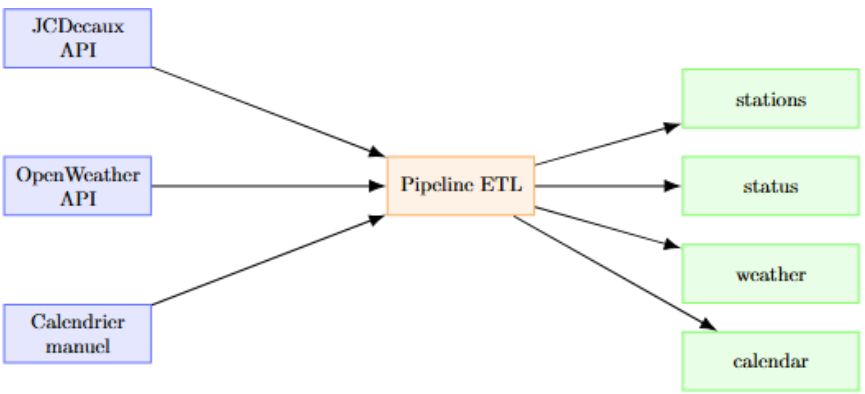
On monitore et gère les erreurs lors du chargement : on annule l'insertion des lignes où il y a une erreur, et on enregistre les tentatives d'insertion et les erreurs.

Enfin, pour détecter les changements de schémas, il faut régulièrement comparer les schémas de la source et du système cibles. On pourrait par exemple mettre en place une alerte lorsqu'un changement est détecté. Dans ce cas, si c'est un petit changement, tel qu'un ajout de colonnes ou la modification du nom d'une colonne, on peut l'automatiser. Dans le cas contraire, l'alerte prévient que le changement est impossible et on doit mettre en place une procédure de migration des données.

3.4) Monitoring de la partie Loading

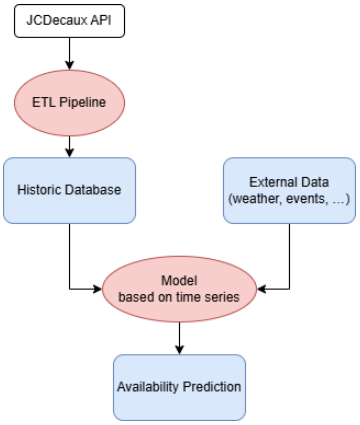
Cela se fait grâce à:
- l'enregistrement de metadata pour chaque chargement dans la BDD
- la vérification des données et le comptage des lignes
- un système d'alerte et de notifications

Finalement, on peut résumer la procédure avec le diagramme suivant:



4) Démonstration

Lien GitHub : https://github.com/guiggzzz/Veflow-Project



Pour la démonstration, nous avons lancé notre pipeline ETL pendant 1h afin de construire une base de données contenant un historique de données de fréquentation des stations. Cette pipeline ETL permet de récupérer les données en temps réel (pas de temps de 1 min) et de les stocker dans une base SQL

La démonstration ne comporte pas la partie monitoring lors du stockage des données. De plus, le code proposé réalise seulement un appel à l'API et stocke les données récupérées. Pour effectuer une historisation automatique, il faudrait exécuter le code dans un environnement Cloud (type Databricks ou Snowflake) pour que le script tourne en continu.

Le but ici est d'obtenir un profil de disponibilité pour les bornes de l'ISAE-Supaéro et de Saint-Pierre.
A l'aide d'un croisement avec des données extérieures et d'un modèle basé sur des séries temporelles nous pourrons prédire les disponibilités de bornes et de vélos pour toutes les stations de Toulouse.
Dans le cadre de cette étude, on se limite à l'obtention de la base de données historique (qui servira de base à l'apprentissage). Une fois le programme terminé, à partir de la base de données obtenue, on peut alors représenter la disponibilité de la station ISAE-Supaero, avec les variations de la quantité de vélos disponibles, sur l'heure étudiée.