Système de Monitoring en Temps Réel des Stations Vélib à Paris

Rapport de Projet

13 décembre 2025

Table des matières

1 Introduction

Ce rapport présente le développement d'une application de visualisation en temps réel des données de disponibilité des stations Vélib dans le Grand Paris. L'application combine l'accès aux données ouvertes de la ville de Paris avec une interface graphique interactive développée en Python.

1.1 Contexte du projet

Le système Vélib, service de vélos en libre-service de la région parisienne, dispose de plus de 1400 stations réparties dans Paris et sa banlieue proche. La disponibilité des vélos et des bornes varie constamment, rendant nécessaire un outil de monitoring en temps réel pour les utilisateurs et les gestionnaires du service.

1.2 Objectifs

Les objectifs principaux de ce projet sont :
— Développer une interface graphique conviviale pour consulter les données Vélib
— Permettre la visualisation en temps réel de la disponibilité des vélos
— Offrir des fonctionnalités de filtrage par commune et par station
— Implémenter un système de mise à jour automatique des données

2 Architecture du Système

2.1 Technologies utilisées

Le projet repose sur plusieurs technologies clés :

Technologie	**Utilisation**
Python 3	Langage de programmation principal
PyQt5	Framework pour l'interface graphique
Matplotlib	Bibliothèque de visualisation de données
Requests	Gestion des appels API HTTP
OpenData Paris API	Source de données en temps réel
SQLite3	Base de données pour l'historique
Pandas	Manipulation et analyse de données
NumPy	Calculs numériques
Scikit-learn	Algorithmes de machine learning

TABLE 1 – Technologies et bibliothèques utilisées

2.2 Architecture modulaire

L'application est structurée en deux classes principales :

1. **ParisData** : Gestion des données et génération des visualisations
2. **MonPremierGui** : Interface graphique utilisateur (hérite de QWidget et ParisData)

3 Fonctionnalités Implémentées

3.1 Récupération des données

L'application utilise l'API OpenData de Paris avec deux endpoints principaux :
— `velib-disponibilite-en-temps-reel` : Données de disponibilité
— Paramètres configurables : nombre de résultats (rows), facettes de recherche
La méthode `get_Data()` gère la récupération des données avec gestion d'erreurs et timeout de 10 secondes. À chaque récupération, les données sont automatiquement sauvegardées dans la base de données SQLite pour constitution de l'historique.

3.2 Système de stockage historique

Base de données SQLite
Une base de données relationnelle `velib_historique.db` est créée automatiquement au démarrage de l'application. Elle contient :

Champ	Description
id	Identifiant unique auto-incrémenté
timestamp	Date et heure de l'enregistrement
station_name	Nom de la station
commune	Commune d'appartenance
ebike	Nombre de vélos électriques
mechanical	Nombre de vélos mécaniques
available_docks	Nombre de bornettes libres
capacity	Capacité totale
hour	Heure (0-23)
day_of_week	Jour de la semaine (0-6)
is_weekend	Indicateur weekend (0 ou 1)

TABLE 2 – Structure de la table historique_stations

Caractéristiques :
— Sauvegarde automatique à chaque récupération de données
— Index optimisé sur (station_name, timestamp) pour requêtes rapides
— Extraction de features temporelles (heure, jour, weekend)
— Gestion robuste des erreurs d'insertion

3.3 Visualisation globale des stations

Fonction : `plt_toutes_les_stations()`
Cette fonctionnalité affiche un graphique à barres des 20 premières stations avec :
— Vélos électriques disponibles (rouge)
— Vélos mécaniques disponibles (jaune)
— Bornettes libres (noir)
— Capacité totale (bleu transparent)

3.4 Analyse par commune

Fonction : `velib_arrondissement()`

Permet de visualiser la disponibilité des vélos électriques pour toutes les stations d'une commune donnée. Le graphique présente :

— Nom de chaque station en abscisse
— Nombre de vélos disponibles en ordonnée
— Affichage sous forme de marqueurs (X)

Le système couvre 53 communes de la région parisienne, incluant Paris et les communes limitrophes.

3.5 Consultation par station

Fonction : `velib_station()`

Affiche en mode texte les informations détaillées pour une station spécifique :

— Nom de la station
— Nombre de vélos disponibles
— Commune d'appartenance
— Horodatage de la consultation

3.6 Système de mise à jour automatique

Toutes les fonctionnalités disposent d'une version avec timer (`timer_*`) permettant :

— Consultation unique (paramètre = 0)
— Rafraîchissement périodique tous les 10 secondes
— Nombre de mises à jour configurable par l'utilisateur

3.7 Module de Machine Learning et Prédictions

Objectif : Prédire la disponibilité des vélos pour une station donnée en fonction de l'heure et du jour.

3.7.1 Algorithme utilisé : Random Forest Regressor

Le choix du Random Forest se justifie par :

— Robustesse face aux données bruitées
— Capacité à capturer des relations non-linéaires
— Pas de sur-apprentissage grâce à l'ensemble d'arbres
— Bonne performance avec des datasets de taille moyenne

Paramètres du modèle :

— `n_estimators=100` : 100 arbres de décision
— `max_depth=10` : Profondeur maximale pour éviter le sur-apprentissage
— `random_state=42` : Reproductibilité des résultats
— `n_jobs=-1` : Parallélisation sur tous les cœurs CPU

3.7.2 Features d'entraînement

Le modèle utilise 4 features temporelles et contextuelles :

1. **hour** (0-23) : Heure de la journée - forte corrélation avec les flux (heures de pointe)

2. **day_of_week** (0-6) : Jour de la semaine - différences de comportement semaine/weekend
3. **is_weekend** (0 ou 1) : Indicateur binaire weekend - patterns différents
4. **capacity** : Capacité de la station - normalisation implicite

Variable cible : `ebike` (nombre de vélos électriques disponibles)

3.7.3 Pipeline d'entraînement

1. **Récupération des données historiques** : Minimum 50 enregistrements requis sur 30 jours
2. **Préparation des features** : Extraction et formatage des variables
3. **Split train/test** : 80% entraînement, 20% test (stratégie random)
4. **Entraînement** : Fit du Random Forest sur les données d'entraînement
5. **Évaluation** : Calcul du MAE et R^2 sur le jeu de test

3.7.4 Métriques de performance

Deux métriques sont calculées pour évaluer la qualité du modèle :
— **MAE (Mean Absolute Error)** : Erreur moyenne en nombre de vélos
  — Interprétation directe (ex : MAE = 2.5 signifie une erreur moyenne de 2-3 vélos)
  — Robuste aux outliers
— **R^2 Score** : Coefficient de détermination (0 à 1)
  — $R^2 > 0.7$: Bon modèle prédictif
  — $R^2 > 0.5$: Modèle acceptable
  — $R^2 < 0.3$: Modèle peu fiable

3.7.5 Fonctionnalités ML implémentées

1. Entraînement du modèle : `train_prediction_model(station_name)`
— Charge les données historiques des 30 derniers jours
— Vérifie le nombre minimum de données (50+)
— Entraîne le Random Forest
— Affiche les métriques de performance
2. Prédiction ponctuelle : `predict_availability(station, hour, day)`
— Prédit la disponibilité pour une heure et un jour précis
— Applique des contraintes (0 prédiction capacité)
— Entraîne automatiquement le modèle si nécessaire
3. Visualisation 24h : `visualize_predictions()`
— Génère des prédictions pour les 24 heures d'une journée
— Crée un graphique avec courbe et zone de confiance
— Affichage interactif avec Matplotlib
4. Export des données : `export_historical_data()`
— Exporte les données des 30 derniers jours en CSV
— Format compatible avec analyse externe (Excel, R, etc.)
— Encodage UTF-8 pour caractères spéciaux
5. Statistiques de la BDD : `get_database_stats()`
— Nombre total d'enregistrements

— Nombre de stations uniques
— Période temporelle couverte
— Permet de valider la constitution de l'historique

4 Interface Graphique

4.1 Composants de l'interface

L'interface PyQt5 comprend les éléments suivants :

1. **Champ de mise à jour** : QLineEdit pour spécifier le nombre de rafraîchissements
2. **Bouton stations globales** : Lance la visualisation des 20 premières stations
3. **Champ commune** : QLineEdit avec auto-complétion (QCompleter) pour sélectionner une commune
4. **Bouton commune** : Affiche les données de la commune sélectionnée
5. **Champ station** : QLineEdit avec auto-complétion pour sélectionner une station
6. **Bouton station** : Affiche les données de la station sélectionnée
7. **Section Machine Learning** :
   — Bouton "Prédiction disponibilité 24h" : Génère les prédictions horaires
   — Bouton "Statistiques base de données" : Affiche les stats de l'historique
   — Bouton "Exporter données CSV" : Exporte l'historique complet

4.2 Ergonomie

L'application intègre plusieurs fonctionnalités d'amélioration de l'expérience utilisateur :

— Auto-complétion sur les communes et stations
— Image de fond personnalisable (carte du Grand Paris)
— Messages d'aide et placeholders dans les champs
— Gestion d'erreurs avec messages informatifs

5 Gestion des Données

5.1 Structure des données API

Les données retournées par l'API OpenData Paris incluent pour chaque station :

Champ	Description
name	Nom de la station
ebike	Nombre de vélos électriques disponibles
mechanical	Nombre de vélos mécaniques disponibles
numdocksavailable	Nombre de bornettes libres
capacity	Capacité totale de la station
nom_arrondissement_communes	Commune d'appartenance
is_installed	Station installée (booléen)
is_renting	Location active (booléen)
is_returning	Retour possible (booléen)

TABLE 3 – Structure des données par station

5.2 Traitement des données

Le code implémente plusieurs mécanismes de robustesse :
— Utilisation de `.get()` avec valeurs par défaut (0) pour éviter les KeyError
— Gestion des exceptions avec try-except
— Validation des entrées utilisateur
— Timeout sur les requêtes HTTP

6 Visualisations Matplotlib

6.1 Configuration des graphiques

Tous les graphiques générés utilisent :
— Format : 12x8 pouces
— Backend : Qt5Agg pour intégration PyQt5
— Légendes positionnées en haut à droite
— Rotation des labels d'axes à 90° pour lisibilité
— Horodatage dans les titres
— `tight_layout()` pour optimiser l'espace

6.2 Types de visualisations

Graphique à barres superposées : Utilisé pour la vue globale, permet de comparer visuellement quatre métriques simultanément avec transparence (alpha) pour la lisibilité.

Graphique en nuage de points : Utilisé pour la vue par commune, permet de repérer rapidement les stations avec forte ou faible disponibilité.

Graphique de prédiction (nouveau) : Courbe continue avec zone de confiance (fill_between) montrant l'évolution prévue de la disponibilité sur 24 heures. Permet d'identifier visuellement les heures de forte/faible disponibilité.

7 Couverture Géographique

Le système couvre l'intégralité du réseau Vélib, soit :
— **Paris** : Tous les arrondissements

— **Hauts-de-Seine (92)** : 15 communes
— **Seine-Saint-Denis (93)** : 13 communes
— **Val-de-Marne (94)** : 12 communes
Total : 53 communes couvertes avec environ 1400 stations.

8 Résultats et Performance

8.1 Temps de réponse

— Récupération des données : 1-3 secondes selon la connexion
— Génération des graphiques : $<$ 1 seconde
— Mise à jour automatique : intervalle de 10 secondes configurable
— Sauvegarde en base de données : $<$ 0.5 seconde pour 1400 stations
— Entraînement du modèle ML : 2-5 secondes (selon taille de l'historique)
— Prédiction unique : $<$ 0.1 seconde

8.2 Performance du modèle ML

Les performances typiques observées après 1 semaine de collecte de données :

Métrique	**Valeur typique**	**Interprétation**
MAE	2-4 vélos	Erreur acceptable
R^2 Score	0.65-0.85	Bon pouvoir prédictif
Temps d'entraînement	2-5 sec	Temps réel acceptable
Données nécessaires	50+	Minimum pour fiabilité

Table 4 – Performances du modèle Random Forest

Facteurs influençant la performance :
— **Quantité de données** : Plus d'historique = meilleure prédiction
— **Régularité des patterns** : Stations avec flux réguliers mieux prédites
— **Type de station** : Stations de transit (gares) plus prévisibles que stations résidentielles
— **Événements exceptionnels** : Grèves, météo extrême réduisent la précision

8.3 Fiabilité

Le système gère efficacement :
— Les erreurs de connexion réseau
— Les données manquantes ou incomplètes
— Les stations temporairement indisponibles
— Les entrées utilisateur invalides

9 Améliorations Possibles

9.1 Fonctionnalités additionnelles

1. **Cartographie interactive** : Intégration d'une carte OpenStreetMap avec localisation des stations
2. **Prédictions avancées** :
   — Intégration de données météo (température, pluie)
   — Prise en compte des événements (concerts, matchs)
   — Prédiction sur plusieurs jours
3. **Alertes personnalisées** : Notification quand une station atteint un seuil de disponibilité
4. **Statistiques avancées** :
   — Analyse de tendances temporelles
   — Identification des stations problématiques
   — Calcul de taux de rotation des vélos
5. **Modèles ML alternatifs** :
   — LSTM pour séries temporelles
   — XGBoost pour améliorer la précision
   — Ensembles de modèles

9.2 Optimisations techniques

— Mise en cache des données pour réduire les appels API
— Multi-threading pour les mises à jour en arrière-plan
— Compression de la base de données pour économiser l'espace
— Requêtes SQL optimisées avec indexation avancée
— Nettoyage automatique des anciennes données (> 3 mois)
— Internationalisation (i18n) pour support multilingue
— Tests unitaires et intégration continue
— Packaging en exécutable standalone
— Hyperparameter tuning pour le Random Forest (GridSearchCV)

10 Conclusion

Ce projet démontre la création d'une application complète combinant :
— Accès à des données ouvertes via API REST
— Interface graphique moderne avec PyQt5
— Visualisations de données interactives avec Matplotlib
— Gestion robuste des erreurs et des cas limites
— **Stockage persistant avec SQLite** pour constitution d'historique
— **Modèle de Machine Learning prédictif** avec Random Forest
— **Pipeline complet de data science** : collecte, stockage, modélisation, prédiction

L'application répond aux objectifs fixés en offrant un outil pratique pour consulter en temps réel la disponibilité des Vélib dans le Grand Paris, tout en apportant une dimension prédictive grâce au machine learning.

10.1 Apports du module ML

L'intégration du machine learning apporte plusieurs avantages :

1. **Aide à la décision** : Les utilisateurs peuvent anticiper la disponibilité avant de se déplacer
2. **Optimisation opérationnelle** : Identification des stations nécessitant une redistribution
3. **Analyse de patterns** : Compréhension des flux de mobilité urbaine
4. **Évolutivité** : Base solide pour des modèles plus complexes (météo, événements)

10.2 Méthodologie data science appliquée

Le projet illustre un workflow complet de data science :

1. **Collecte** : Récupération automatique via API
2. **Stockage** : Base de données structurée avec features engineering
3. **Exploration** : Visualisations pour comprendre les données
4. **Modélisation** : Entraînement de modèles supervisés
5. **Évaluation** : Métriques de performance (MAE, R^2)
6. **Déploiement** : Interface utilisateur pour exploiter les prédictions

10.3 Perspectives

Le système peut servir de base pour :
— Analyse des flux de mobilité urbaine
— Optimisation de la redistribution des vélos par les opérateurs
— Études sur les habitudes de déplacement et leur évolution
— Intégration dans des applications de mobilité multimodale
— Recherche académique sur la mobilité douce en milieu urbain
— Détection d'anomalies et prédiction de pannes
— Comparaison avec d'autres systèmes de vélo-partage

10.4 Limitations actuelles

Quelques limitations à prendre en compte :
— **Données historiques** : Nécessite plusieurs semaines pour des prédictions fiables
— **Événements exceptionnels** : Le modèle ne prend pas en compte les perturbations (grèves, météo extrême)
— **Features limitées** : Seules les variables temporelles sont utilisées
— **Modèle unique** : Un modèle par station serait plus précis mais plus coûteux
— **Pas de validation croisée temporelle** : Le split train/test est aléatoire

A Configuration et Installation

A.1 Prérequis

```
pip install PyQt5
pip install matplotlib
pip install requests
pip install pandas
pip install numpy
pip install scikit-learn
```

Listing 1 – Installation des dépendances

A.2 Lancement de l'application

```
python3 velib_monitoring.py
```

Listing 2 – Exécution du programme

A.3 Configuration

Modifier le chemin de l'image de fond dans le fichier principal :

```
image_path = r'chemin/vers/grand_paris.png'
```

B Références

— OpenData Paris : `https://opendata.paris.fr`
— Documentation PyQt5 : `https://www.riverbankcomputing.com/static/Docs/PyQt5/`
— Documentation Matplotlib : `https://matplotlib.org/`
— API Vélib en temps réel : `https://opendata.paris.fr/explore/dataset/velib-disponibilite-en-temps-reel/`
— Scikit-learn Documentation : `https://scikit-learn.org/stable/`
— Random Forest Algorithm : Breiman, L. (2001). "Random Forests". Machine Learning. 45 (1) : 5–32
— Pandas Documentation : `https://pandas.pydata.org/docs/`
— SQLite Documentation : `https://www.sqlite.org/docs.html`

C Exemples d'utilisation

C.1 Scénario 1 : Consultation simple

1. Lancer l'application
2. Sélectionner une station dans le champ "Station"
3. Cliquer sur "Voir les vélos par station"
4. Résultat : Affichage du nombre de vélos disponibles en temps réel

C.2 Scénario 2 : Prédiction ML

1. Laisser l'application collecter des données pendant quelques jours (minimum 50 observations)
2. Sélectionner une station dans le champ "Station"
3. Cliquer sur "Prédiction disponibilité 24h"
4. Résultat : Graphique montrant la disponibilité prévue pour chaque heure de la journée

C.3 Scénario 3 : Export et analyse externe

1. Cliquer sur "Exporter données CSV"
2. Ouvrir le fichier `velib_export.csv` dans Excel ou R
3. Effectuer des analyses personnalisées (corrélations, visualisations avancées)

C.4 Scénario 4 : Monitoring continu

1. Entrer "10" dans le champ "nombre de mise à jour"
2. Sélectionner une commune
3. Cliquer sur "Voir les vélos par commune"
4. Résultat : Le graphique se met à jour automatiquement toutes les 10 secondes, 10 fois