Cahier des charges

# **Projet ML Ops** : Prédiction du temps de réponse des pompiers à Londres

## Contexte et Objectifs

La London Fire Brigade (LFB) est le plus grand corps de sapeurs-pompiers dans le monde avec 103 casernes et plus de 5000 sapeurs-pompiers professionnels. La Brigade couvre les 13 circonscriptions de Londres qui est composé de 8 millions d’habitants. La Brigade traite entre 100000 et 130000 appels d’urgences chaque année et intervient sur un territoire de 1587m².

En cas d’urgence vitale, toute minute perdue amoindrit les chances de survie ou augmente le risque de séquelles. Côté incendie, un fameux dicton le dit : la première minute, un feu s’éteint avec un verre d’eau, la deuxième avec un seau, et la troisième avec une citerne ! Il est donc primordial que les pompiers interviennent le plus rapidement possible sur le site. Dans ce contexte, il serait intéressant pour la brigade, et notamment pour le centre d’appels de pouvoir anticiper et prédire le temps d’intervention des pompiers suite à un appel. Cela permettrait notamment de rassurer les personnes appelant les secours, de mieux optimiser la logistique des forces de secours ou de challenger l’organisation du départ et les trajets pour réduire encore le temps de prise en charge de l’urgence.

Problématique :

Lors d’un appel d’urgence, pouvoir prédire en temps réel avec quelques informations récoltées par téléphone le temps que mettra le premier camion de pompiers pour arriver sur les lieux.

Commanditaire :

La direction des services opérationnels de la London Fire Brigade

Utilisateur :

L’équipe de régulation en charge de la réception des appels (Une centaine de personnes se relaient pour la réception des appels d'urgence). Chaque utilisateur doit être en mesure d’obtenir une prédiction du temps d’arrivée des forces de pompiers suite à un appel d’urgence après avoir entré quelques informations. Les informations à renseigner doivent être le plus simple possible à obtenir par téléphone.

Administrateur :

L’administrateur doit être en mesure :

- d’obtenir les métriques de performance du modèle

- de réentrainer le modèle sur les nouvelles données

- d’obtenir la liste des features présentent dans la database

- d’obtenir un échantillon de la database

Dans quel contexte l’application devra-t-elle s’intégrer :

L’application doit pouvoir être exploitée via l’application interne déjà utilisée par le call center de la London Fire Brigade. Cette solution a l’avantage d’offrir une possibilité de connexion simple a logiciel existant, de ne pas alourdir le nombre d’applications utilisées par les membres du call center, et de ne pas perturber leurs habitudes de travail.

Via quel support l’application sera-t-elle utilisée :

L’application sera sous forme de deux API : « user » et « admin ».

Dans le cadre de notre projet nous mettrons en place une application avec interface graphique à l’aide de streamlit. Ainsi l’application streamlit fera des requêtes sur les différentes routes de l’API User, ce qui permettra d’illustrer ce à quoi devra ressembler le résultat final une fois la solution déployée.

## Modèle

Il s’agit d’un modèle de régression dont l’objectif est de prédire au plus proche lle temps d’intervention des pompiers suite à un appel d’urgence.

Features :

* IncidentGroup object
* StopCodeDescription object
* PropertyCategory object
* IncGeo\_BoroughCode object
* IncGeo\_WardCode object
* Easting\_rounded int64
* Northing\_rounded int64
* IncidentStationGround object
* FirstPumpArriving\_AttendanceTime float64
* FirstPumpArriving\_DeployedFromStation object
* NumStationsWithPumpsAttending int32
* NumPumpsAttending int32
* PumpMinutesRounded int64
* NumCalls int32
* PumpOrder int64
* DelayCodeId int32
* Year int32
* DayOfWeek int32
* Hour int32

Variable cible :

FirstPumpArriving\_AttendanceTime (temps en secondes entre le moment de l’appel d’urgence et le moment d’arrivée des premiers secours sur les lieux)

Algorithmes utilisés :

LightGBM Regressor (Gradient Boosting)

Random Forest Regressor

Performance du modèle :

Afin d’assurer un modèle performant en production, nous souhaitons obtenir une erreur moyenne inférieure à 2min (soit 120 secondes) sur le temps de prédiction.

Sur nos deux modèles retenus, l’objectif est atteint et dépasse nos attentes en atteingnant une erreur moyenne entre 1min et 1min30.

* LGBM :
  + Mean Absolute Error (MAE): 63.300027560231136
  + R-squared (R²): 0.6423356903564763
  + Root Mean Squared Error (RMSE): 88.16856456447708
* Random Forest :
  + Mean Absolute Error (MAE): 58.07854019532286
  + R-squared (R²): 0.6639329304360693
  + Root Mean Squared Error (RMSE): 85.77518899409165

Métriques d’évaluations :

Nous choisissons d’observer principalement la RMSE pour évaluer la performance de notre modèle, car celle-ci est facilement interprétable et tient compte des valeurs extrèmes. Ici nous avons une RMSE de 86.33, soit environ 1 minute et 26 secondes de différence moyenne entre la prédiction du temps d’intervention et le temps réel.

## Base de données :

Les données utilisées :

* Données sur les incidents**:** <https://data.london.gov.uk/dataset/london-fire-brigade-incident-records>
* Données sur les forces de la Brigade qui ont été mobilisés suite à l’incident : <https://data.london.gov.uk/dataset/london-fire-brigade-mobilisation-records>

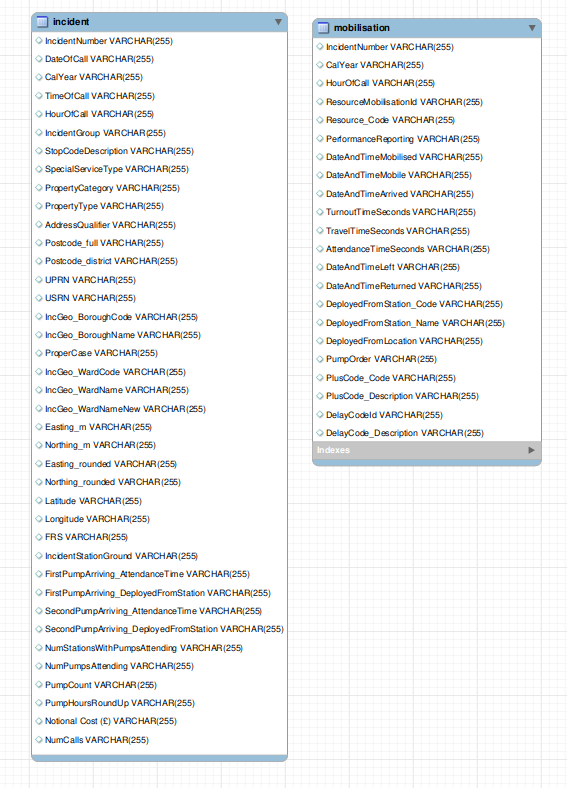
Outil de gestion de base de données :

Nous avons choisis d’utiliser MySQL pour la gestion de notre base de donnée.

Pour des raisons pratiques et d’accessibilité de l’application, nous avons choisi d’héberger notre base de donnée dans le cloud.

Pour cela nous avons opté pour le cloud service RDS de Amazon AWS. Ce service nous a semblé le plus adapté car il s’adapte parfaitement à des données structurées et gérées via MySQL ou PostgreSQL.

Architecture de la base de données :



Gestion des nouvelles données :

Les nouvelles données ne sont pas ajoutées au fil de l’eau mais sont chargées en bloc tous les trimestres. En effet ces nouvelles données sont mises à disposition en libre accès par le London Datastore tous les 3 mois.

Nous avons pris le parti de stocker de manière brut ces données sans les transformer. En effet, la transformation se fera via les scripts python après requête. Il s’agit d’une solution ELT.

## API

Cette API assure le lien entre la base de données, le modèle entraîné et l’utilisateur.

Outil utilisé : FastAPI

Liste des fonctionnalités de l’API User :

* Authentification
  + Vérification que l’utilisateur apparatient bien à la liste des Users prédéfinis.
* Model
  + Obtenir une prédiction à partir de certaines données d’entrée

Liste des fonctionnalités de l’API Admin :

* Authentification
  + Vérification que l’utilisateur apparatient bien à la liste des Admin prédéfinis.
* Vérifier le bon fonctionnement de l’API
* Base de données :
  + Vérifier la structure de la base de donnée
  + Obtenir un échantillon de la base de donnée
* Machine Learning :
  + Metrics: Obtenir les métriques d’un modèle donné déjà entrainé
  + Training: Permet d’entrainer un modèle sur les données présentes dans la database à l’instant T.

## Isolation et portabilité

Nous avons choisi d’utiliser Docker pour containeriser notre application. Chacune des deux API devra être dans un container docker distinct, et qui renverront vers des ports distinct définis afin d’être toutes deux disponibles en simultannée.

Ces containers docker sont déployés via un Docker Compose.

## Testing & Monitoring

Nos tests sont réalisés via PyTest. Ils sont directement intégrés au workflow de GitHub Actions, ainsi chaque merge ou pull request lancera automatiquement toute

Liste des tests unitaires à mettre en œuvre :

* Test de connexion à la base de donnée OK
* Test du bon format des données importées depuis la base de donnée
* Test des métriques du modèle actuellement en production
* Test des différentes routes de l’API Admin
* Test des différentes routes de l’API User

Monitoring :

La LFB fournit chaque mois les nouvelles données concernant les interventions du mois précédent.

Après chaque nouvelle entrée mensuelle, le modèle devra s’évaluer pour vérifier les métriques. Si la RMSE du modèle dépasse le seuil de **120** alors on procède à un nouvel entrainement du modèle sur les nouvelles données présentes en base.

Nous automatisons l’évaluation et l’entrainement du modèle en continue à l’aide de Airflow et de MLFlow :

1. Charger les données
2. Créer le dataset
3. Entrainer les modèles
4. Évaluer les modèles
5. Sélectionner le meilleur modèle
6. Exporter le meilleur modèle

A noter que nos modèles sont enregistrés sur le cloud d’Amazon AWS à l’aide d’un service S3 Bucket.

## Schéma d’implémentation

