# MLOPS

Mlops vient de l’anglais Machine Learning Operations; il s’agit d’un cycle d’étapes permettant de normaliser la mise en production des modèles de Machine Learning puis d’assurer leur maintenance et supervision de façon autant automatisée que possible.

Voici ces différentes étapes et quelques outils open-source permettant de les effectuer.

* **Analyse exploratoire des données (EDA)**: Chaque set de données va être versionné et conservé en-ligne pour garder leur historique et permettre de les partager et utiliser facilement.   
  Une première exploration de ces sets est effectuée à ce niveau. Elle va permettre de qualifier leur distribution, leur valeurs aberrantes et manquantes, si des données personnelles soumises au GDRP sont présentes et si en fin de compte ces données sont de vraiment utiles pour nos besoins business et de bonne qualité. Des visualisations peuvent également être produites et partagées à ce stade pour aider à cette tâche.
  + *DVC* (https://dvc.org/) est un bon outil open-source qui permet entre autre de gérer ces versions de datasets en sauvant leur metadata sur git (les données restent sur le cloud, par exemple dans un bucket S3 d’AWS). *Datachain* mentioné sur la même page permet de créer des pipelines qui permettront d’automatiser l’exploration et visualisation des données, voire également leur préparation et ingénierie des fonctionnalités.
  + Pour l’exploration et visualisation des données l’outil *Ydata profiling* (<https://github.com/ydataai/ydata-profiling>, anciennement appelé *pandas-profiling*) permet d’analyser facilement des set de données complexes ou d’en comparer avec peu de code. D’autres outils similaires plus spécialisés dans la visualisation sont *SweetViz* (<https://github.com/fbdesignpro/sweetviz>) et *Dtale* (<https://github.com/man-group/dtale>).
* **Préparation des données et ingénierie des fonctionnalités**: Cette étape permet de préparer les données pour être utilisée lors de l’entraînement des modèles. A ce niveau des fonctionnalités vont être sélectionnées et transformées (comme par exemple une réduction de dimensions ou du word embeddings comme durant ce projet), de nouvelles fonctionnalités seront générées si besoin est et certaines fonctionnalités seront normalisées ou mises à l’échelle. Ces processus vont être automatisés dans le pipeline MLOps.
  + D’autres bons outils pour créer des pipelines recoupant au moins ces premières étapes sont *Kedro* (<https://kedro.org/>) et *Pachyderm* (la version community est open-source: <https://www.pachyderm.com/>). Ces fonctions de préparation et ingénierie des fonctionnalités seront généralement écrites en Python et intégrées dans ces pipelines directement.
  + *DVC* mentionné dans l’étape précédente ou un autre outil similaire comme *lakeFS* (https://lakefs.io/) peuvent également être utilisés ici pour stocker ces sets de données transformés et enrichis.
  + On peut aussi utiliser une *feature store*, une base de données spécialisée qui permet de stocker, centraliser, partager et gérer efficacement les données des fonctionnalités. Elles facilitent également le réutilisage des fonctionnalités entre différentes équipes, réduit les efforts de duplication et aide à maintenir la qualité et la traçabilité des données. De bons exemples open-source sont *Feast* (<https://github.com/feast-dev/feast>) et *FeatureForm* (<https://github.com/featureform/featureform>).

**Entraînement et ajustement des modèles:** Une fois que les fonctionnalités sont prêtes le modèle peut être entraîné grâce à ces dernières. Différents (hyper)paramètres du modèle vont être testés afin de déterminer quelle combinaison va produire les meilleurs résultats en fonction des besoins business.

* + La librairie open-source *scikit-learn* est un grand classique pour entraîner et optimiser différents modèles. La librairie *Hyperopt* (<https://hyperopt.github.io/hyperopt/>) peut également être utilisée pour cette étape.
* **Examen et gouvernance des modèles**: Durant cette étape les modèles et leur metadata (version, configuration, documentation, différents métriques d’évaluation, variables d’environnement, …) vont être sauvés dans un registre qui va permettre de suivre leur évolution, de les comparer et d’obtenir une bonne visibilité sur chacun tout en les partageant à qui de droit.
  + *MLFlow* (<https://mlflow.org/>) est un bon outil introduit dans ce projet qui permet d’accomplir ces goals. *Comet ML* est similaire et a une version gratuite pour la communauté (<https://www.comet.com/site/>).
* **Déploiement et inférence des modèles**: Cette étape concerne le déploiement des modèles pour leur utilisation (inférence). Les modèles peuvent être déployés dans différents environments (dev, test, production) localement ou en-ligne et vont être accessibles via une interface web (REST) ou autre.
  + Un pipeline CI/CD usuel peut être utilisé pour déployer une application web intégrant le modèle et permettant son inférence, typiquement un service Flask en Python déployé sur le cloud comme pour ce projet.
  + Des librairies existent pour encapsuler un ou plusieurs modèles et leur fournissent une interface pour l’inférence comme *BentoML* (<https://github.com/bentoml/BentoML>).
  + Les plateformes cloud courantes (AWS, Azure, Google cloud, …) offrent un vaste éventail de services ML permettant notamment leur hébergement sur leur plateforme. Utiliser ces plateformes permet également d’intégrer différentes étapes du MLOps directement sur leur plateforme en utilisant leurs différents outils propriétaires, cela facilite la tâche même si l’on finira fortement couplé à leur plateforme.
* **Surveillance des modèles**: Les modèles déployés seront monitorés via différents métriques pour s’assurer qu’ils marchent correctement et que les prédictions des modèles sont toujours précises.  
  Les métriques techniques permettent d’apprécier l’état des services déployés en terme de ressources utilisés, requêtes reçues et répondues, … afin de s’assurer que tout marche correctement et que par exemple il ne faut pas attribuer plus de ressources au service ou démarrer une nouvelle instance (*horizontal/vertical scaling*).  
  Une variation des données (de type *data drift* quand la distribution des données changent ou *concept drift* dû à un environnement mouvant) peut engendrer des prédictions erronées des modèles et dans ce cas ces derniers doivent être ré-entraînés avec des données à jour pour corriger leurs prédictions. Le processus de détection de ces variation et du ré-entraînement et déploiement des modèles à jour peut être automatisé dans le cadre du MLOps.  
  Dans le cadre de ce projet un *data drift* proviendrait par exemple du fait que les utilisateurs posent des questions de plus en plus depuis leur téléphone mobile, impliquant des questions plus courtes et concises et cela pourrait perturber notre modèle. Une variation de type *concept drift* proviendrait par exemple du fait que les thèmes des questions changent avec le temps, typiquement quand de nouveaux langages de programmation sont introduits pendant que d’anciens sont abandonnés ou par exemple si de nouveaux JS frameworks deviennent très populaires.   
  + Les métriques techniques peuvent être collectés par ou envoyés à un outil tel que *Prometheus* (<https://prometheus.io/>) qui va les sauver et exposer pour analyse ou visualisation par un outil tel que *Grafana* (<https://grafana.com/>). Certains métriques de performance ML peuvent également être stockés dans *Prometheus*.
  + Les plateformes Cloud ont leur propre monitoring en place pour ces métriques et peuvent réagir à leur variation avec par exemple des alertes ou changement du système (plus ou moins d’instances par exemple).
  + Une bonne méthode pour détecter une variation des données est de conserver toutes les nouvelles données reçue par un service d’inférence et leurs prédictions effectuées par le modèle (comme dans ce projet) pour pouvoir les examiner et valider par la suite.   
    Si une variation est détectée les nouvelles données peuvent être utilisées avec les données originelles (par exemple) pour entraîner un nouveau modèle qui sera déployé en production automatiquement. Une notification devrait également être envoyée pour alerter les personnes responsables.
  + Une librairie telle que *Frouros* (<https://github.com/IFCA-Advanced-Computing/frouros>) permet d’effectuer ce genre d’analyse basée sur différents métriques (par exemple un test Kolmogorov-Smirnov). Ces métriques pourraient être générés régulièrement et envoyés sur *Prometheus* par exemple pour pouvoir les visualiser facilement sur un dashboard *Grafana* pour permettre aux équipes concernées de déterminer quand ré-entraîner le modèle.
  + Des librairies plus complètes permettent également de produire des rapports et offre des dashboards de monitoring par défaut, comme par exemple *Evidently AI* (<https://github.com/evidentlyai/evidently>).
  + Dans le cadre de ce projet un script pourraient être lancé chaque mois sur AWS Lambda téléchargeant les questions du mois dernier via l’API et utilisant le dernier modèle stocké sur S3 afin de déterminer une quelconque déviation via par exemple la librairie Frouros. A la place des questions du dernier mois le script pourrait également utiliser les questions envoyées au service et leur prédiction (stocké en db et accessible via un endpoint REST).   
    Si les métriques d’évaluation dépasseraient un seuil prédéfini une notification serait lancée et un script d’entraînement d’un nouveau modèle commencerait en utilisant les questions depuis 2010 (par exemple) jusqu’à la date d’exécution du script. Une fois entraîné l’ancien modèle serait archivé, le modèle à jour serait téléversé sur S3 et les services d’inférence seraient redémarrés automatiquement afin d’utiliser ce nouveau modèle.