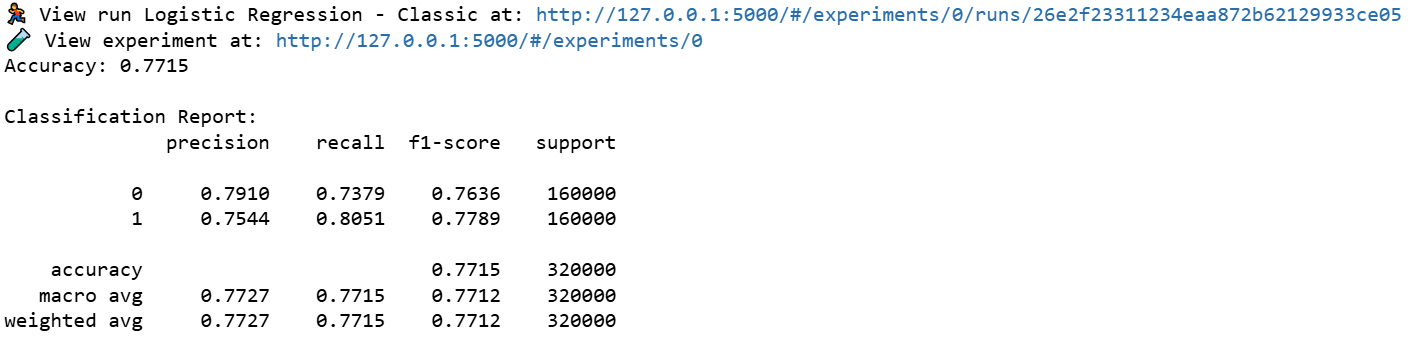
**Introduction**

Dans cet article, nous présentons un projet de prédiction de sentiment à partir de tweets. Ce projet inclut plusieurs approches de modélisation et met en œuvre des principes MLOps pour assurer la gestion efficace des expérimentations, la reproductibilité des résultats, et le déploiement d'un modèle en production.

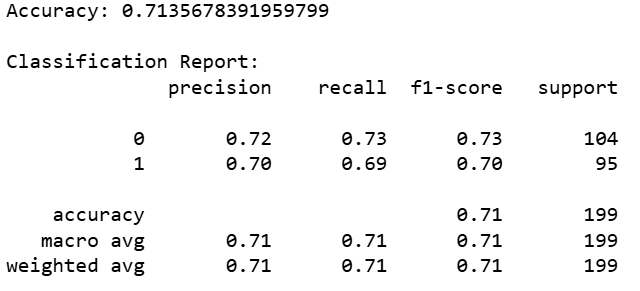
**1. Comparaison des Trois Approches**

1. **Modèle classique** :
   * La régression logistique est utilisée pour la prédiction des sentiments avec les embeddings générés par BERT. Cela est correctement implémenté dans le premier bloc de code, où le modèle de régression logistique est entraîné sur les embeddings BERT, et des évaluations comme l'accuracy et le classification report sont fournies.



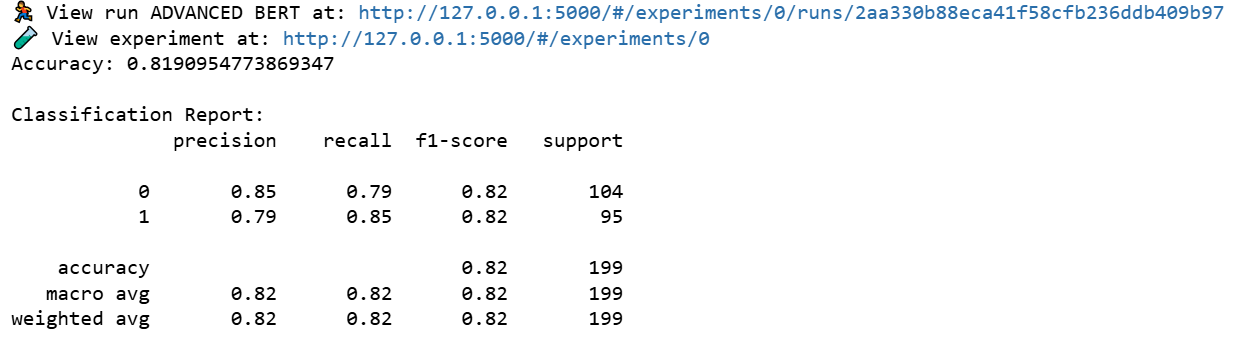
* + **Complet** : Oui, cette approche est couverte avec l'utilisation de LogisticRegression pour la classification des sentiments basée sur les embeddings.

1. **Modèle sur mesure avancé** :
   * Ce modèle est abordé avec l'utilisation des **embeddings BERT** pour entraîner un modèle de régression logistique, ce qui peut être considéré comme un modèle avancé dans cette approche.



* + Le modèle est également suivi et loggé dans **MLflow**, et la performance est enregistrée dans des graphiques et rapports.
  + **Complet** : Oui, cette approche est correctement couverte en utilisant BERT pour les embeddings et en enregistrant l'expérience via MLflow.

1. **Modèle avancé BERT** :
   * Le code contient également une section où un modèle Keras est créé et entraîné avec des embeddings BERT (par exemple, en utilisant une architecture avec des couches Dense et Dropout), ce qui pourrait être interprété comme un "modèle avancé" qui utilise des embeddings de BERT pour la classification des sentiments.



* + Les détails de l'entraînement et de l'évaluation du modèle sont fournis, ainsi que la sauvegarde du modèle dans **MLflow**.
  + **Complet** : Oui, cette approche est couverte avec un réseau de neurones en utilisant les embeddings BERT pour la classification.

**Points supplémentaires :**

* **MLOps (MLflow)** : Le script inclut l'intégration de **MLflow** pour le suivi des expérimentations (log des modèles, des paramètres, des métriques et des artefacts comme les graphiques, matrices de confusion, etc.).
* **Évaluation** : L'évaluation des modèles (régression logistique et Keras) est réalisée avec les métriques classiques : accuracy, classification report, roc\_curve, et confusion\_matrix. Ces évaluations sont également loggées dans MLflow pour une traçabilité.
* **Sauvegarde des modèles** : Les modèles sont sauvegardés dans un répertoire spécifique et également loggés dans **MLflow**.

**Comparaison des Modèles de Prédiction de Sentiments**

| **Modèle** | **Accuracy** | **Précision (Classe 0 / 1)** | **Recall (Classe 0 / 1)** | **F1-score (Classe 0 / 1)** |
| --- | --- | --- | --- | --- |
| **Régression Logistique (Classique)** | **0.7715** | 0.7910 / 0.7544 | 0.7379 / 0.8051 | 0.7636 / 0.7789 |
| **Modèle Sur-Mesure Avancé** | **0.7136** | 0.72 / 0.70 | 0.73 / 0.69 | 0.73 / 0.70 |
| **Modèle Avancé BERT** | **0.8191** | 0.85 / 0.79 | 0.79 / 0.85 | 0.82 / 0.82 |

**1. Régression Logistique (Modèle Classique)**

* **Performance :** Il affiche une **accuracy de 77.15%** et un bon équilibre entre **précision et recall**.
* **Forces :** Facile à entraîner, rapide à exécuter et interprétable. Convient aux jeux de données de grande taille
* **Faiblesses :** Peut être limité pour capturer les relations complexes entre les mots et leur contexte dans un texte.

**2. Modèle Sur-Mesure Avancé**

* **Performance :** Moins performant que la régression logistique avec **71.36% de précision**, malgré un bon équilibre entre les classes.
* **Forces :** Peut inclure des embeddings spécifiques ou des architectures neuronales personnalisées.
* **Faiblesses :** Faible performance ,ce qui pourrait être dû à un sous-ajustement ou un manque de données d'entraînement.

**3. Modèle Avancé BERT**

* **Performance :** Le modèle le plus performant avec **81.91% d’accuracy**, un f1-score équilibré de **0.82**, et une meilleure généralisation.
* **Forces :** Exploite la puissance des **transformers et word embeddings contextuels** pour mieux comprendre le sens des mots et des phrases.
* **Faiblesses :** Temps d'entraînement et de prédiction plus longs, consommation élevée en ressources GPU/TPU.

**Conclusion**

* **Si la rapidité et l’interprétabilité sont prioritaires**, la **régression logistique** est un bon choix.
* **Si on cherche à capturer plus de subtilités textuelles tout en restant léger**, le **modèle sur-mesure avancé** peut être une alternative, mais nécessite une amélioration.
* **Si la performance est le critère principal**, le **modèle BERT** est **le plus performant**, malgré son coût élevé.

**2. Démarche MLOps**

**2.1 Suivi des expérimentations avec MLFlow**

MLFlow est un outil clé pour la gestion des expérimentations dans ce projet. Il permet de suivre toutes les expérimentations, qu'il s'agisse de modèles classiques ou avancés, de comparer leurs performances et de centraliser les résultats. Les hyperparamètres utilisés, les métriques de performance et les modèles sont sauvegardés dans un répertoire central, ce qui facilite le versioning et le suivi des résultats au fil du temps.

L'interface MLflow Tracking est composée de plusieurs sections permettant de suivre et d'analyser les expérimentations des modèles de Machine Learning.

**📌 Structure de l'interface MLflow**

**1️⃣ Page d'accueil (Experiments Tracking)**

La page d'accueil affiche la liste des expériences et des exécutions (runs) enregistrées. Chaque ligne correspond à une exécution avec des informations clés telles que :

✅ Le nom de l'expérience associée

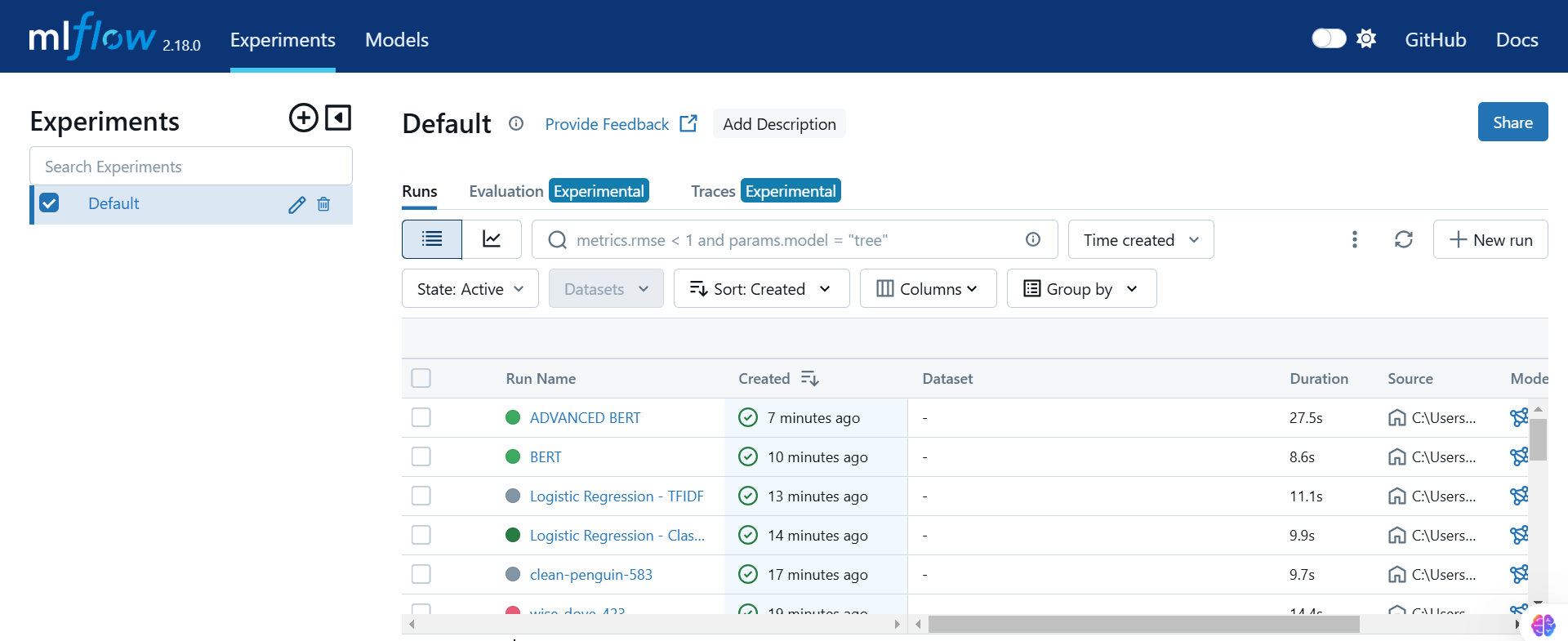
✅ Un ID unique pour chaque exécution

✅ Les hyperparamètres enregistrés (log\_param)

✅ Les métriques principales (log\_metric, ex. : accuracy, F1-score, loss, etc.)

✅ L’état du run (succès ou échec)

✅ Le lien vers les artefacts (plots, modèles sauvegardés, etc.)



En cliquant sur un run spécifique, on accède aux détails de l'expérimentation.

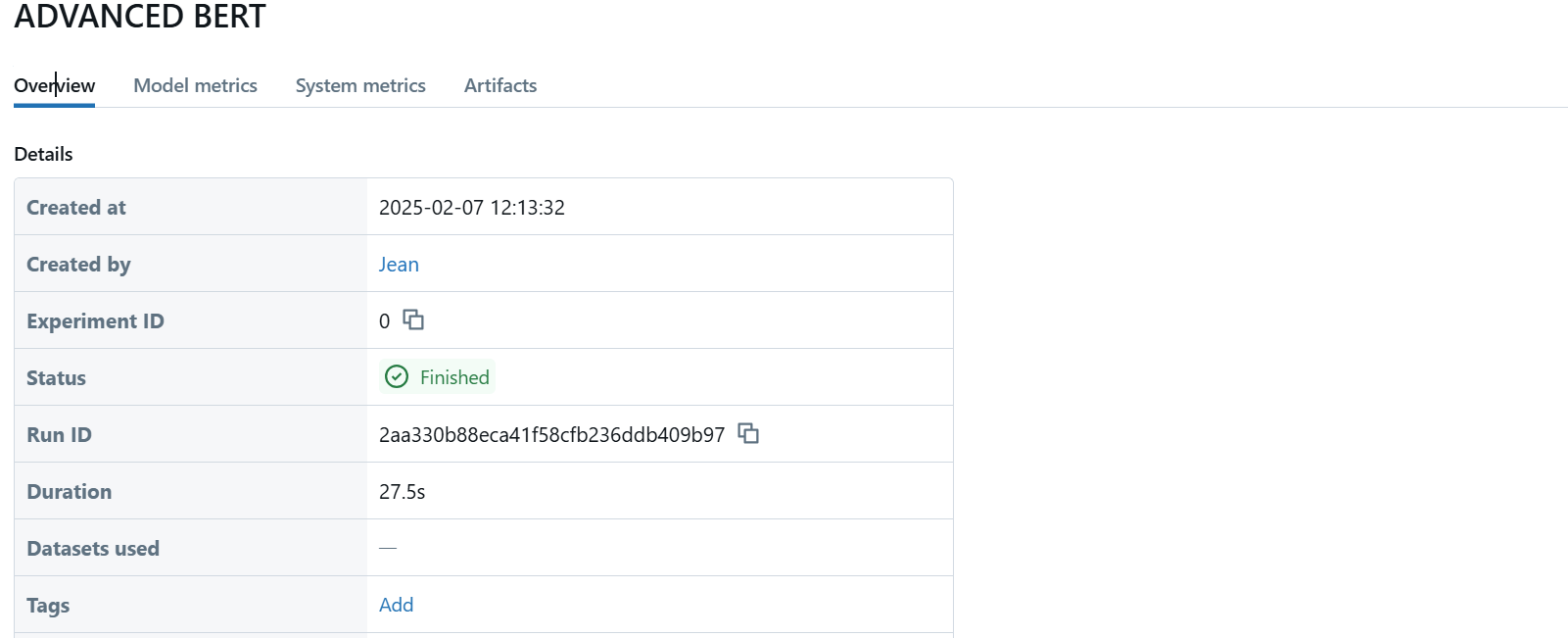
**2️⃣ Page d’un modèle spécifique**

Après avoir sélectionné un modèle dans la liste, on accède à une page dédiée avec plusieurs onglets :

🔷 **Overview (Vue d'ensemble)**

Présente une description du modèle, ses paramètres, et ses caractéristiques (ex. : algorithme utilisé, date d'entraînement, version du framework comme scikit-learn==1.2.0 ou TensorFlow==2.10).

Affiche un lien vers le modèle sauvegardé, permettant un téléchargement ou un chargement pour réutilisation (mlflow.sklearn.load\_model).



🔷 **Model Metrics (Métriques du modèle)**

Contient toutes les métriques enregistrées durant l’entraînement (mlflow.log\_metric).

Exemples de métriques affichées :

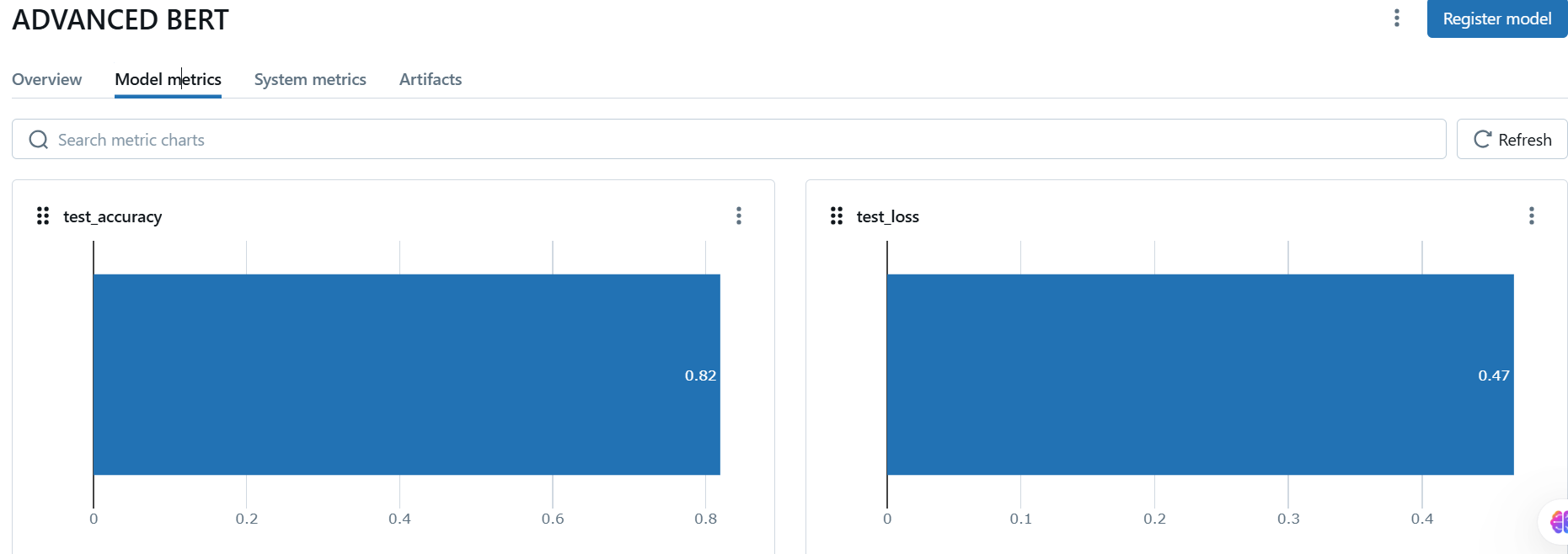
Accuracy : Score global de précision du modèle.

Precision, Recall, F1-score : Indicateurs de performance pour chaque classe.

Loss (perte) : Indicateur de l’erreur du modèle.

AUC-ROC, Log-Loss, RMSE selon le type de tâche.

Les métriques sont affichées sous forme de tableau et de graphiques pour une analyse visuelle rapide.



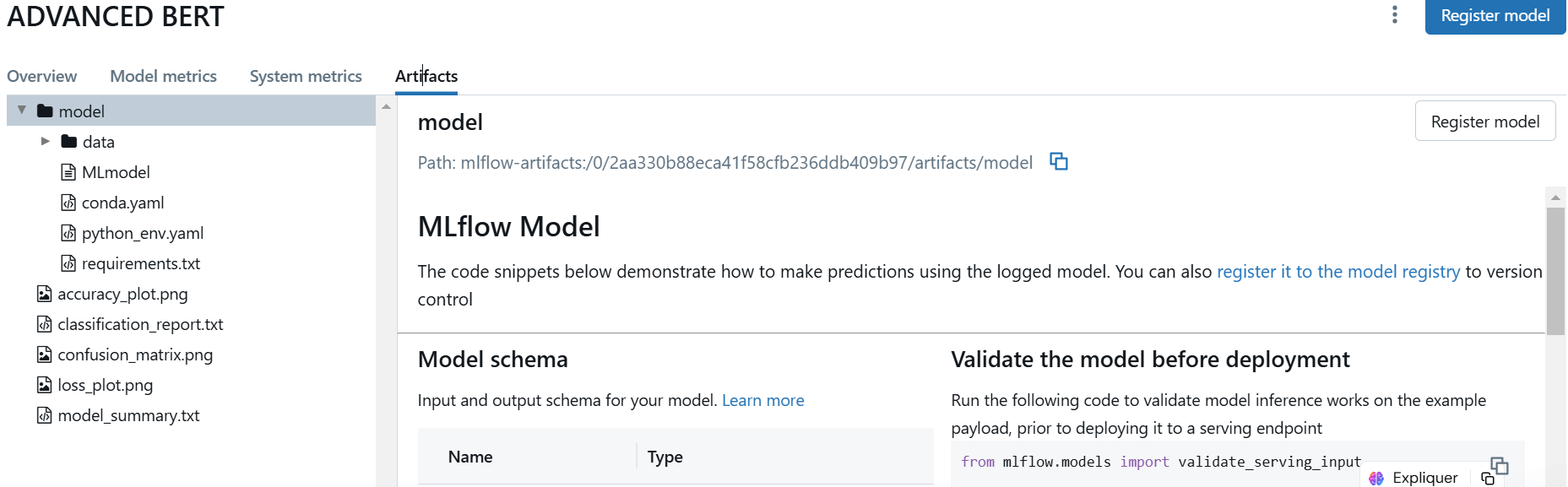
🔷 **Artifacts (Fichiers et Plots sauvegardés)**

Regroupe tous les fichiers enregistrés (mlflow.log\_artifact), notamment :

📊 **Graphiques d'évaluation** : matrice de confusion, courbe ROC, histogramme des probabilités, etc.

🔢 Modèle sauvegardé : fichier model.pkl, MLmodel, ou model.onnx pour réutilisation.

📜 **Logs et fichiers supplémentaires** : rapports d'expérimentation, fichiers CSV des prédictions, etc.



🔷 **Comparaison des runs**

MLflow permet de comparer plusieurs exécutions en superposant leurs métriques et artefacts.

Cela facilite l'analyse des performances pour déterminer le modèle optimal.

**2.2 Déploiement et gestion des versions**

Les modèles entraînés sont déployés sous forme d'API via **AWS Elastic Beanstalk**. L'API permet de recevoir des tweets et de renvoyer les prédictions en temps réel. Un pipeline CI/CD (intégration continue et déploiement continu) a été mis en place pour tester et déployer automatiquement les mises à jour du modèle dans l'environnement de production, garantissant ainsi une gestion fiable des versions du code.

**2.3 Tests unitaires et validation continue**

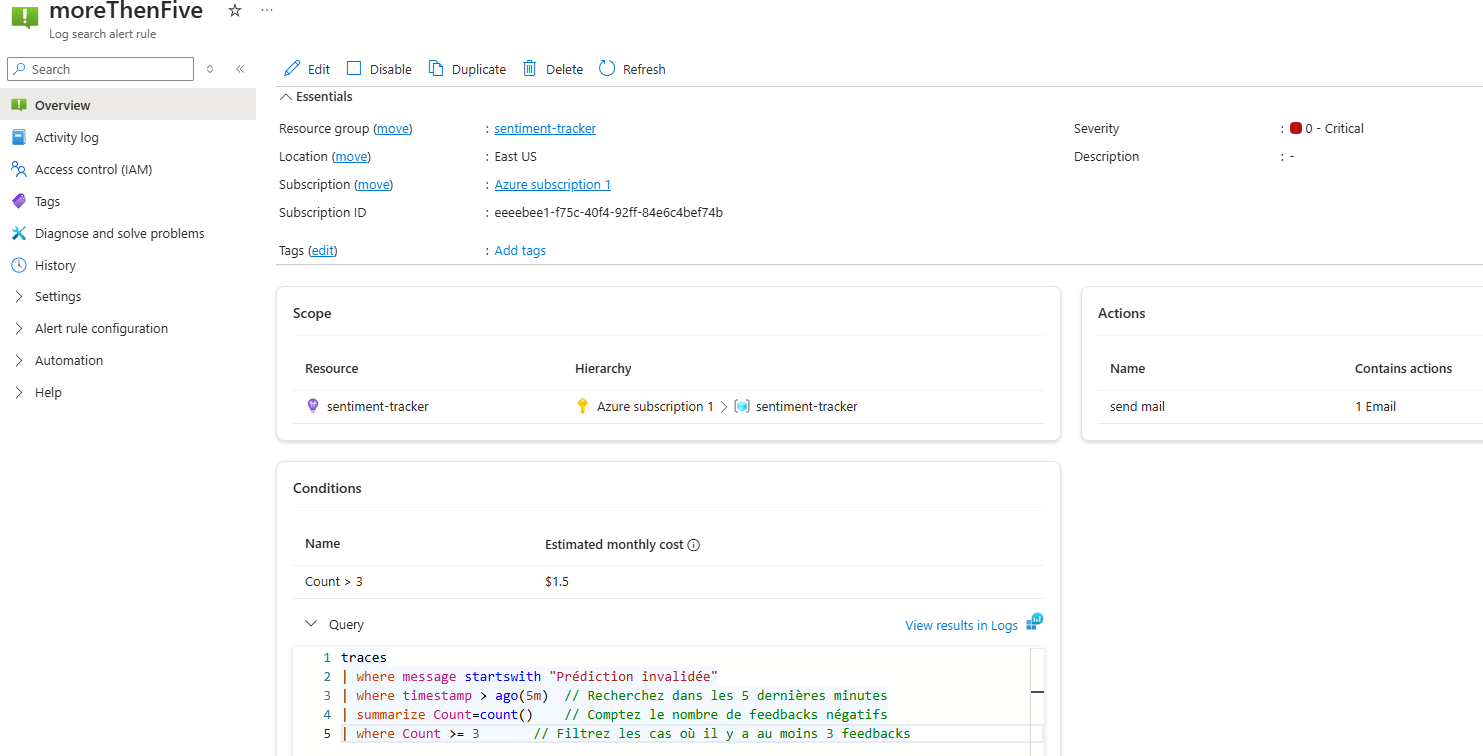
Des tests unitaires sont intégrés dans le pipeline via **pytest**. Ces tests vérifient que l'API fonctionne correctement et que le modèle déployé produit des prédictions valides. Cela permet de s'assurer que les changements apportés au code ou au modèle n'introduisent pas de régressions.

**3. Suivi de la Performance en Production**

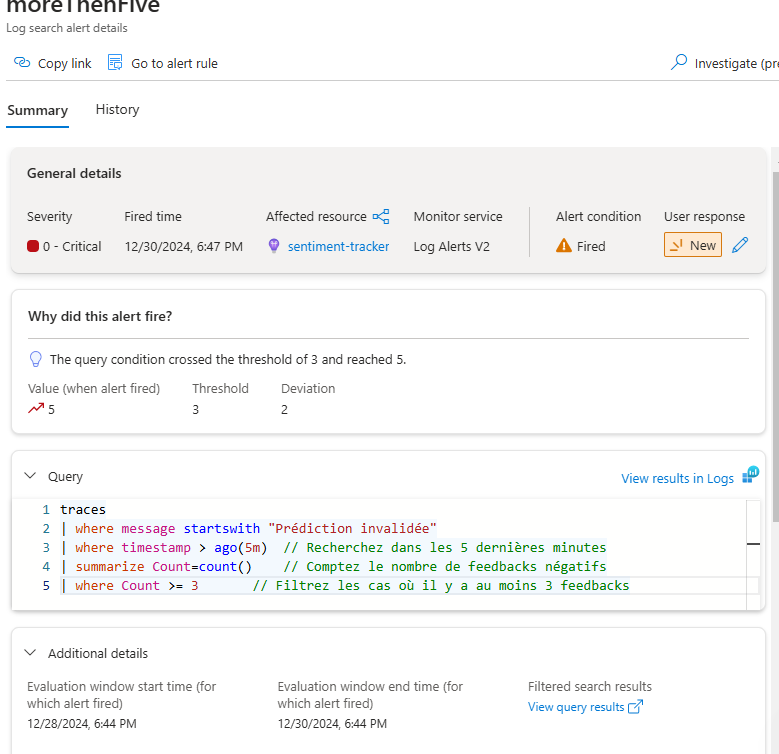
**Suivi des Performances en Production et Amélioration Continue du Modèle**

Dans le cadre du suivi de la performance du modèle en production, nous avons mis en place un système de monitoring via **Azure Application Insights**, permettant de détecter les anomalies et d’alerter en cas de problème. Ce système repose sur la collecte de **traces** et l’envoi d’**alertes automatiques**, assurant une supervision efficace et une réactivité rapide face aux éventuels dysfonctionnements du modèle de prédiction des sentiments.

**3.1. Détection des anomalies et envoi d’alertes**



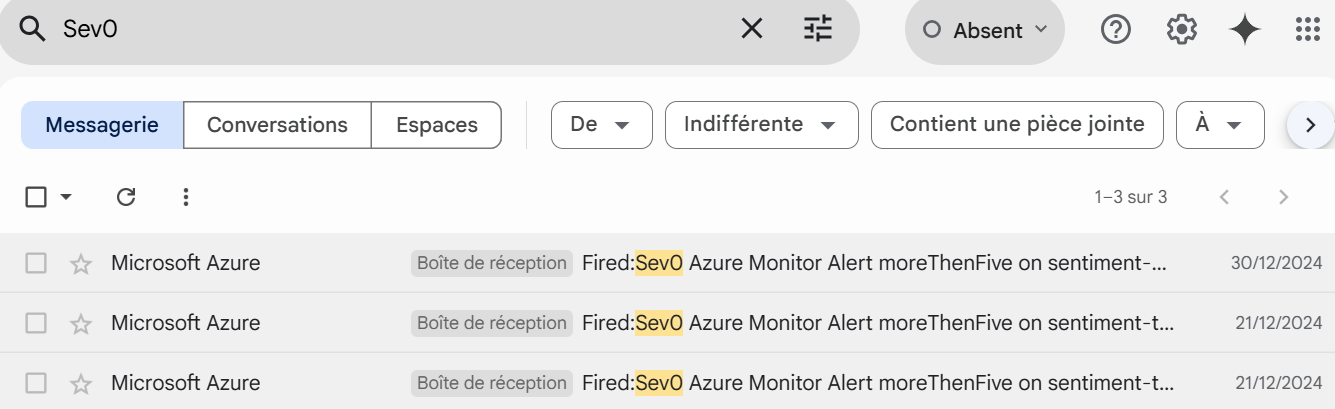
La première capture d’écran illustre l’overview de l’alerte configuré dans **Azure Application Insights**. Cette alerte est déclenchée lorsque certaines conditions spécifiques sont remplies, telles qu’un taux d’erreur élevé dans les prédictions ou une latence excessive des requêtes API.



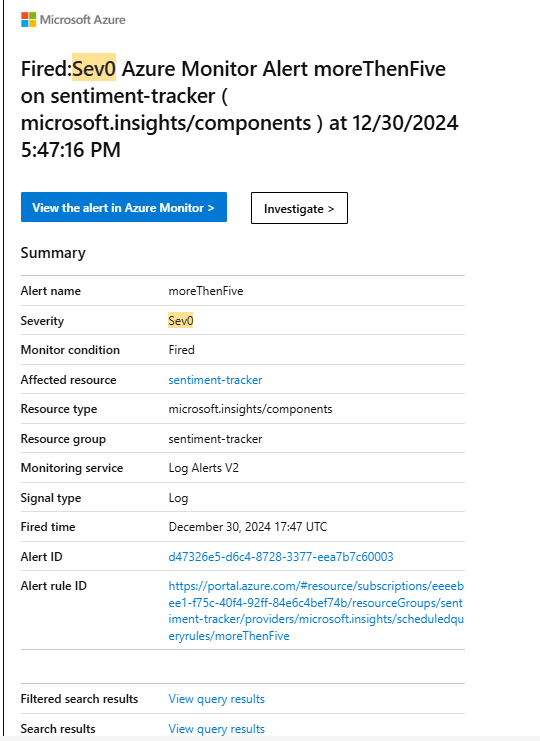
La deuxième capture détaille le **résumé d’une alerte spécifique**, comprenant la requête associée et les métriques ayant déclenché cette alerte. Cela permet aux équipes techniques de diagnostiquer précisément l’origine du problème et d’y remédier efficacement.

**3.2. Notifications et gestion proactive des incidents**

Pour assurer une réactivité immédiate, chaque alerte génère une **notification par e-mail** aux équipes concernées.



Cette capture montre **l’ensemble des e-mails reçus** en réponse aux alertes déclenchées. Ce suivi centralisé des notifications garantit que chaque incident est bien pris en compte et traité rapidement.



La dernière capture illustre **le contenu d’un e-mail d’alerte**, précisant la nature de l’incident, son niveau de gravité et les actions recommandées. Grâce à ces informations détaillées, les équipes peuvent intervenir efficacement pour minimiser l’impact sur la production.

**3. Analyse des statistiques et amélioration continue du modèle**

Afin d’exploiter ces alertes dans une démarche d’amélioration continue, nous proposons une méthodologie basée sur les étapes suivantes :

1. **Analyse des tendances des alertes** : Identification des types d’erreurs récurrents (ex. : tweets mal classifiés, ralentissement de l’API).
2. **Corrélation avec les données d’entrée** : Étude des cas problématiques pour détecter des biais dans les données ou des limitations du modèle.
3. **Optimisation du modèle** : Ajustement des hyperparamètres, exploration de nouvelles architectures ou enrichissement des données d'entraînement.
4. **Déploiement d’un correctif** : Mise à jour du modèle et suivi des améliorations via **Azure Application Insights** pour valider les effets des modifications.

Cette approche permet d’assurer une qualité de service optimale et d’adapter le modèle aux évolutions des données et des besoins métiers.

**Conclusion**

Ce projet de prédiction de sentiment utilise trois approches différentes pour la modélisation, allant d'une solution simple avec régression logistique à une solution plus complexe avec BERT. En appliquant des principes MLOps, le projet s'assure que les expérimentations sont bien gérées, que les modèles sont versionnés et que le déploiement en production est fiable. Le suivi de la performance via Azure Application Insights permet de maintenir un haut niveau de qualité et d'améliorer continuellement le modèle.