

APPROCHES ET OUTILS POUR UNE DÉMARCHE ORIENTÉE MLOPS

1). Pipeline de codage des étapes d'élaboration du modèle

a). Collecte des données via une requête SQL

Nous partons du principe que si l'outil est adopté par Stack Overflow, nous pourrions requêter directement leur base de données, sans avoir à passer par leur page publique [StackExchange](#).

b). Vérification des données

- Mise en place d'une fonction qui permettra de vérifier que la structure des données est toujours identique et que les données seront bien adaptées pour la suite du pipeline.
- Si nécessaire, mise en place d'une fonction qui écarte toutes données personnelles ou non pertinentes pour la suite des traitements.

c). Préparation des données

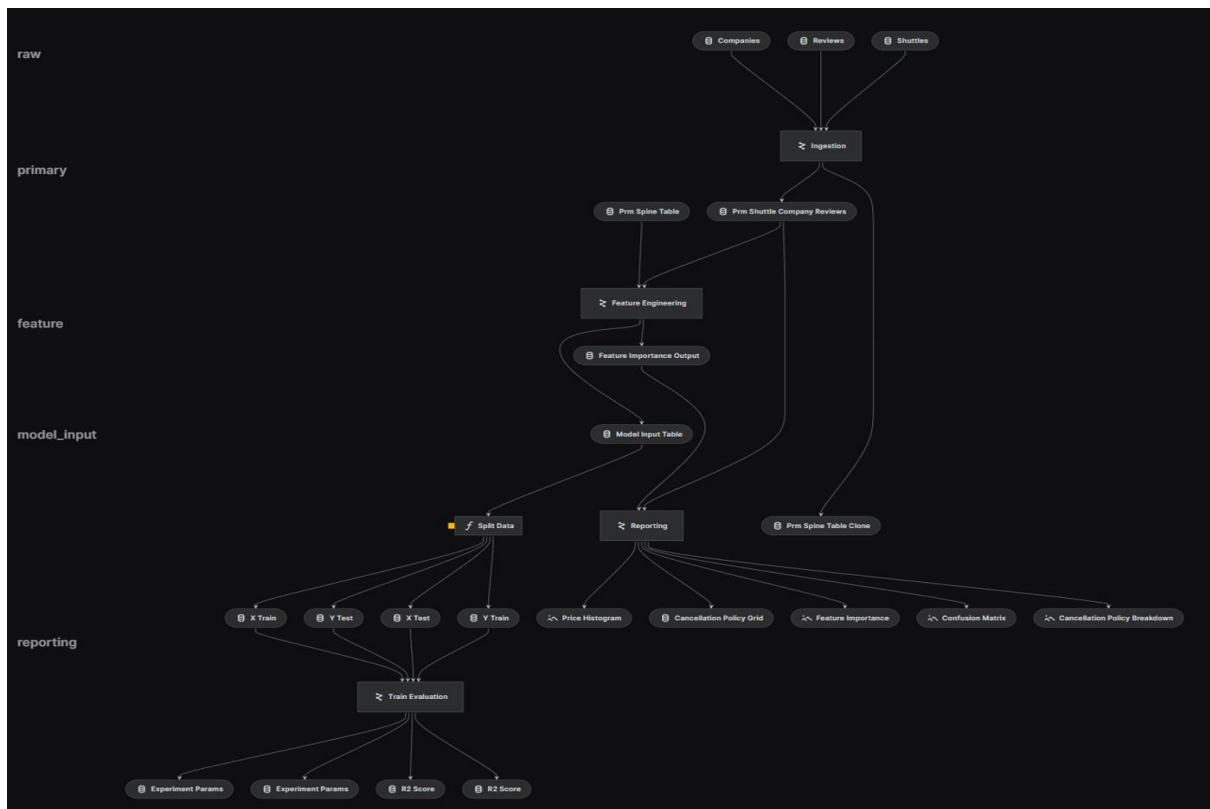
Puisque notre modèle de sentence embedding que nous utilisons est [USE](#) (Universal Sentence Encoder), les prétraitements des données textuelles seront peu nombreux :

- Fonction de concaténation de *Title* et *Body*.
- Fonction de suppression des documents vides.
- Fonction de suppression des doublons.
- Fonction de suppression des balises HTML.
- Fonction de suppression des portions de code.
- Fonction d'encodage des documents avec USE.
- Fonction d'encodage des tags associés avec *MultiLabelBinarizer*.

d). Entraînement du modèle

- Entraînement du modèle avec séparation train/test.
- Log des métriques classiques et métier.
- Si les métriques sont satisfaisantes, enregistrer le modèle entraîné dans le registre.

Ces étapes pourront être implémentées à l'aide de [Kedro](#). Il s'agit d'un framework open-source développé pour faciliter et améliorer la productivité des projets de data science et d'apprentissage automatique. Il vise à rendre le processus de développement de pipelines de données plus fluide et plus organisé en fournissant une structure claire et modulaire pour la création de projets de données. Il est doté d'une interface claire permettant de bien visualiser les étapes du pipeline et de paramétrer et visualiser le code des fonctions utilisées.



Code block

```

def split_data(data: pd.DataFrame, split_options: Dict) -> Tuple:
    """Splits data into features and targets training and test sets.

    Args:
        data: Data containing features and target.
        parameters: Parameters defined in parameters.yml.

    Returns:
        Split data.
    """
    target_variable = split_options["target"]
    independent_variables = [x for x in data.columns if x != target_variable]
    test_size = split_options["test_size"]
    random_state = split_options["random_state"]

    logger = logging.getLogger(__name__)
    logger.info(
        f"Splitting data for the following independent variables "
        f"{independent_variables} against the target of '{target_variable}' "
        f"with a test sized of {test_size} and a random state of "
        f"{random_state}"
    )

    X = data[independent_variables]
    y = data[target_variable]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state
    )

    return X_train, X_test, y_train, y_test

```

Show Code

×

Split Data

Original node name: split_data

Type: node

File Path: ...rc/demo_project/pipelines/modelling/nodes.py

Parameters: { 1 item }
 > "split_options" : { ... } 3 items

Inputs: Model Input Table, Params: Split Options

Outputs: X Train, X Test, Y Train, Y Test

Tags: -

Run Command: Please provide a name argument for this node in order to see a run command.

Les parties log des métriques et enregistrement du modèle dans un registre pourront également être réalisées avec [ML Flow](#).

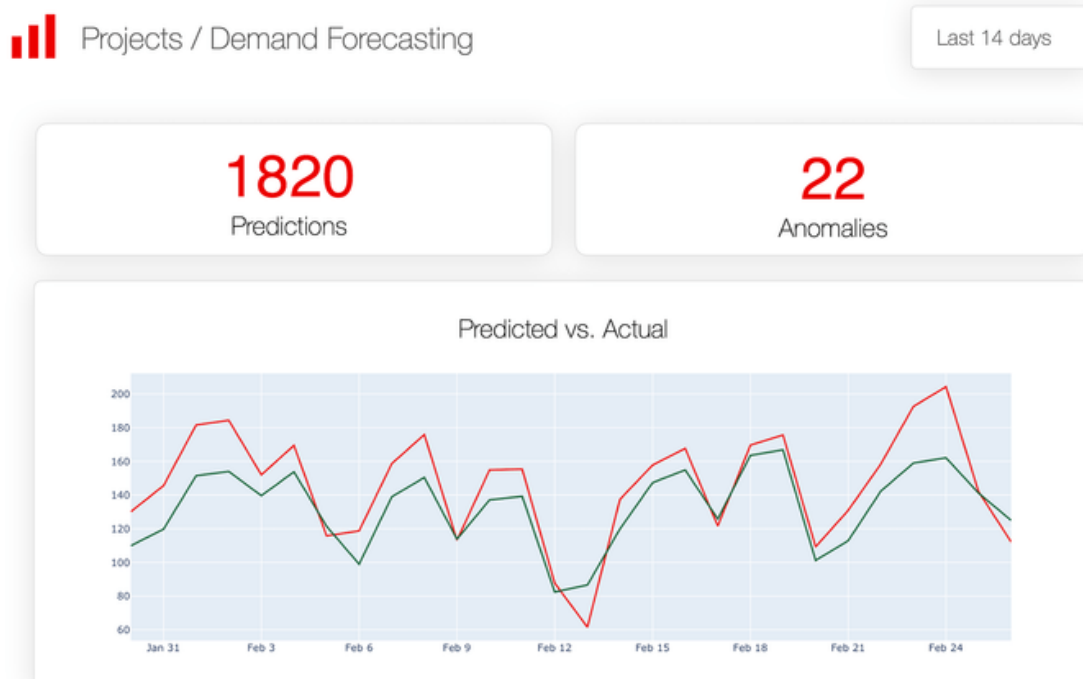
2). Suivi de l'utilisation et de la performance en production

a). Model Drift, Data Drift et Concept Drift

Il s'agit de concepts qui se rapportent à la dégradation de la qualité des résultats fournis par le modèle au cours du temps.

- **Model Drift** : avec le temps, le modèle semble moins performant, en termes de métriques classiques (accuracy, etc.) ou métier.

Le suivi de ces métriques peut être réalisé avec l'outil [Evidently AI](#).











Si la nature des données en entrée semble correcte, le model drift peut notamment provenir de deux phénomènes : le **Data Drift** et le **Concept Drift**.

- **Data Drift** : avec le temps, les données sur lequel s'exécute le modèle peuvent différer de façon importante des données d'entraînement. Il en résultera alors un changement dans la qualité des prédictions. Cette baisse de qualité ne sera pas forcément rapidement détectée avec des métriques classiques, puisque s'agissant de données actuelles, nous n'aurons pas d'étiquette pour vérifier la qualité des prédictions. Il est donc important de détecter des changements dans les données d'entrées.

L'outil [Evidently AI](#) permet de détecter automatiquement un data drift dans les données à l'aide de tests mathématiques.

Drift is detected for 100.0% of columns (5 out of 5).

Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test
> target	cat			Detected	chi-square p_value
> sepal width (cm)	num			Detected	K-S p_value
> sepal length (cm)	num			Detected	K-S p_value
> petal length (cm)	num			Detected	K-S p_value

- **Concept drift** : la relation qui lie les données d'entrée et de sortie, que le modèle avait réussi à modéliser, a changé. Même si les données d'entrée peuvent sembler similaires, les valeurs à prédire n'y sont plus liées de la même manière. Il peut donc être nécessaire de régler de nouveau les hyperparamètres du modèle lors de l'entraînement, voire d'implémenter un nouveau modèle.

3). Système de suivi de la performance

Il est nécessaire de suivre les performances du modèle lors de deux étapes : lors de l'entraînement et en production.

- **Entraînement** : utilisation des métriques classiques (accuracy, precision, recall, F1, etc.) et métier (un tag réel trouvé, tous les tags réels trouvés, taux de couverture, documents sans prédiction).

Pour le suivi de ces performances, il est possible d'utiliser [Evidently AI](#) ou [ML Flow](#).

- **En production** : il est possible de tester le modèle, sans le réentraîner, en lui faisant faire des prédictions sur des données passées mais postérieures à l'entraînement. Nous pourrions suivre les métriques classiques et métier grâce aux outils cités précédemment. Il est également possible de mettre en place des métriques spécifiques pour le modèle en production, par exemple : si les tags prédits sont simplement suggérés aux utilisateurs, à quelles fréquences sont-ils acceptés et utilisés par ces derniers ? À quelle fréquence les utilisateurs ajoutent des tags qui n'ont pas été prédits, voire qui n'existent pas dans l'ensemble des tags que le modèle est capable de prédire ?

Il faudra alors récupérer ces métriques et mettre en place un tableau de bord avec [Evidently AI](#) pour les suivre et éventuellement paramétrer des alertes si certains seuils sont atteints (avec l'envoi d'un e-mail à l'administrateur par exemple). Il faudra aussi intégrer la détection avec [Evidently AI](#) d'un data drift qui pourrait nécessiter de réentraîner le modèle.