**📂 Project Structure with Comments**

Here is your **final project structure**, with **detailed comments** explaining each folder's purpose.

📂 **Listing contents of:** D:\Data\_Science\Append\_Data\_Engineer\_AWS\_MLOPS\Data\_Scientist\_Rakuten\_Project-main

bash

CopierModifier

📄 LICENSE

📄 README.md

📂 data/ # 📌 Contains all datasets (raw, processed, and intermediate)

│ 📂 interim/ # 📌 Intermediate data (cleaned but not final)

│ │ 📄 X\_full\_train.pkl # Merged intermediate training dataset

│ │ 📄 X\_test.pkl # Intermediate test dataset

│ │ 📄 X\_test\_cleaned.pkl # Cleaned version of test data

│ │ 📄 X\_test\_img.pkl # Intermediate test dataset (images metadata)

│ │ 📄 X\_train\_cleaned.pkl # Cleaned version of training data

│ │ 📄 X\_train\_img.pkl # Intermediate training dataset (images metadata)

│ │ 📄 X\_train\_labeled.pkl # Cleaned training dataset with labels

│ │ 📄 y\_train.pkl # Target labels for training

│ 📂 processed/ # 📌 Final datasets ready for modeling

│ │ 📄 X\_test\_final.pkl # Final test dataset (combined text + images)

│ │ 📄 X\_train\_final.pkl # Final training dataset (combined text + images)

│ │ 📂 text/ # 📌 Processed text data

│ │ │ 📄 Xtest\_matrix.pkl # TF-IDF matrix for test data

│ │ │ 📄 Xtrain\_matrix.pkl # TF-IDF matrix for training data

│ │ │ 📄 prdtypecode\_mapping.csv # Mapping of product type codes

│ │ │ 📄 prdtypecode\_mapping.pkl # Pickle version of the mapping

│ │ │ 📄 tfidf\_vectorizer.pkl # Saved TF-IDF vectorizer

│ │ │ 📄 y\_train\_encoded.pkl # Encoded target labels for text

│ │ 📂 images/ # 📌 Processed image data

│ │ │ 📂 resized/ # 📌 Resized images (ready for CNNs or other models)

│ │ │ │ 📂 image\_train/ # Resized training images

│ │ │ │ │ 🖼️ image\_001.jpg

│ │ │ │ │ 🖼️ image\_002.jpg

│ │ │ │ │ 🖼️ image\_003.jpg

│ │ │ │ │ ...

│ │ │ │ 📂 image\_test/ # Resized test images

│ │ │ │ │ 🖼️ image\_101.jpg

│ │ │ │ │ 🖼️ image\_102.jpg

│ │ │ │ │ 🖼️ image\_103.jpg

│ │ │ │ │ ...

│ │ │ 📂 features/ # 📌 Extracted image features (for ML models)

│ │ │ │ 📄 cnn\_features\_train.pkl # CNN-extracted features for training

│ │ │ │ 📄 cnn\_features\_test.pkl # CNN-extracted features for testing

│ │ │ │ 📄 hog\_features.pkl # Features extracted using HOG

│ │ │ │ 📄 sift\_features.pkl # Features extracted using SIFT

│ │ │ │ 📄 pca\_images.pkl # PCA components for image dimensionality reduction

│ │ │ │ 📄 y\_train\_images.pkl # Encoded target labels for images

│ │ 📄 y\_train\_final.pkl # Final target labels (combined text + images)

│ 📂 raw\_csv/ # 📌 Raw CSV files containing text and metadata

│ │ 📄 X\_test\_update.csv # Raw test dataset (text + metadata)

│ │ 📄 X\_train\_update.csv # Raw training dataset (text + metadata)

│ │ 📄 Y\_train\_CVw08PX.csv # Raw training labels

│ 📂 raw\_images/ # 📌 Raw image data (original images)

│ │ 📂 image\_test/ # Unprocessed test images

│ │ │ 🖼️ image\_1000095714\_product\_345301179.jpg

│ │ │ 🖼️ image\_1000139370\_product\_353636820.jpg

│ │ │ 🖼️ image\_1000221680\_product\_353626686.jpg

│ │ │ ...

│ │ 📂 image\_train/ # Unprocessed training images

│ │ │ 🖼️ image\_1000076039\_product\_580161.jpg

│ │ │ 🖼️ image\_1000089455\_product\_348990858.jpg

│ │ │ 🖼️ image\_1000092894\_product\_353108104.jpg

│ │ │ ...

📂 models/ # 📌 Trained models and saved weights

📂 notebooks/ # 📌 Jupyter notebooks for data exploration, preprocessing, and modeling

│ 📂 eda\_and\_processing/ # Exploratory Data Analysis & Preprocessing

│ 📂 modeling/ # Machine learning models

│ │ 📂 bimodal/ # Multi-modal models combining text and image

│ │ │ 📄 7\_Model\_Combination.ipynb

│ │ 📂 image/ # Image modeling notebooks

│ │ │ 📄 6\_Image\_Modeling.ipynb

│ │ 📂 text/ # Text modeling notebooks

│ │ │ 📄 6\_Text\_Modeling.ipynb

│ │ │ 📄 7\_ML\_Text\_Classification.ipynb

│ │ │ 📄 7\_ML\_Text\_Classification\_V00.ipynb

│ │ │ 📄 DL\_RNN\_GRU\_CELL.ipynb

│ │ │ 📄 DL\_RNN\_Conv1D.ipynb

│ │ │ 📄 DL\_RNN\_LSTM\_CELL.ipynb

│ │ │ 📄 DL\_Simple\_DNN.ipynb

│ │ │ 📄 Machine\_learning\_model\_and\_Parameters.ipynb

│ │ │ 📄 Tooba\_ML\_Logistic\_SVM\_RandomForest\_Voting\_XGBOOST\_LinearSVC.ipynb

│ │ │ 📄 iTFIDF\_matrix.ipynb

│ 📄 utils\_remplace\_Variable\_name.ipynb

📂 references/ # 📌 Research papers, documentation, and external resources

📂 reports/ # 📌 Project reports and figures

│ 📂 figures/ # 📌 Visualizations, plots, and charts

│ 📄 final\_report.md # 📌 Final technical report

📄 requirements.txt # 📌 Python dependencies for the project

📂 src/ # 📌 Python scripts for data processing and model training

│ 📂 app/ # Application scripts (e.g., Streamlit, Flask API)

│ 📂 data\_acquisition/ # Scripts for data fetching and ingestion

│ 📂 data\_preprocessing/ # Data cleaning and transformation scripts

│ 📂 model\_combination/ # Fusion of text and image models

│ 📂 modeling\_image/ # Image model training scripts

│ 📂 modeling\_text/ # Text model training scripts

📂 utilis/ # 📌 Utility scripts

│ 📄 Structure\_repository.txt # Repository structure documentation

│ 📄 create\_DS\_project\_structure.py # Script to generate project structure

│ 📄 directory\_tree\_last.py # Last version of the directory listing script

│ 📄 jupyter\_notebook\_config.py.lnk # Jupyter Notebook config

│ 📄 launch\_jupyter.bat # Script to launch Jupyter Notebook

**✅ Why is this version better?**

✔ **Every folder and file has an English description**  
✔ **Clearly separates raw, processed, and model-ready data**  
✔ **Easy to navigate for both text and image processing**  
✔ **Respects best practices for data science and machine learning pipelines**

📌 **Do you need a Python script to move images to the new structure automatically?** 😊🚀

**Vous avez dit :**

cette organisation sera bien dans l'optique de monter mon projet sur cloud AWS avec Cloudformation cdk depuis mon pc en local?

**ChatGPT a dit :**

Oui, **cette organisation est très bien adaptée** pour **déployer ton projet sur AWS avec CloudFormation et AWS CDK**, mais quelques **ajustements** pourraient l’optimiser pour un **déploiement cloud natif**.

**✅ Pourquoi cette organisation est adaptée au cloud AWS ?**

1. **Séparation claire des données brutes, intermédiaires et finales**
   * raw\_csv/ et raw\_images/ → Idéal pour être stocké dans **Amazon S3**.
   * processed/ → Peut être **synchronisé avec un bucket S3** pour être utilisé par les instances d’entraînement (SageMaker, EC2).
2. **Structure modulaire avec src/ et models/**
   * src/ contient les scripts Python bien organisés pour **prétraiter les données et entraîner les modèles**.
   * models/ peut stocker les **modèles entraînés (TensorFlow, PyTorch, Scikit-Learn)** et être sauvegardé dans **S3** ou **SageMaker Model Registry**.
3. **Séparation des notebooks (notebooks/) et du code exécutable (src/)**
   * Les notebooks sont utiles en local, mais dans AWS, **on exécute surtout les scripts Python en batch**.
   * AWS SageMaker Notebooks pourrait être utilisé pour exécuter les notebooks/.
4. **Compatible avec un pipeline ML sur AWS**
   * data/ → Peut être stocké dans **S3**.
   * src/modeling\_image/ et src/modeling\_text/ → Peut être exécuté sur **SageMaker Training Jobs**.
   * reports/ → Peut être stocké dans **S3 ou Amazon QuickSight** pour des visualisations.

**📌 Optimisations pour le cloud AWS avec CloudFormation/CDK**

Voici quelques **ajustements pour une meilleure intégration AWS** :

**1️⃣ Définir un Bucket S3 pour stocker les données**

Au lieu de stocker les datasets localement (data/), on peut les envoyer dans **Amazon S3** :

* raw\_csv/ → **S3://my-bucket/raw\_csv/**
* raw\_images/ → **S3://my-bucket/raw\_images/**
* processed/ → **S3://my-bucket/processed/**

📌 **Comment automatiser l'upload vers S3 ?**  
Ajoute un script upload\_to\_s3.py dans src/data\_acquisition/ :

python

CopierModifier

import boto3

import os

s3 = boto3.client('s3')

bucket\_name = "my-bucket"

def upload\_directory(local\_directory, s3\_prefix):

"""Uploads all files in a local directory to S3 under the specified prefix"""

for root, \_, files in os.walk(local\_directory):

for file in files:

local\_path = os.path.join(root, file)

relative\_path = os.path.relpath(local\_path, local\_directory)

s3\_path = f"{s3\_prefix}/{relative\_path}"

s3.upload\_file(local\_path, bucket\_name, s3\_path)

print(f"Uploaded {local\_path} to s3://{bucket\_name}/{s3\_path}")

# Example: Upload raw data

upload\_directory("../data/raw\_csv", "raw\_csv")

upload\_directory("../data/raw\_images", "raw\_images")

✅ **Avantages :** Automatiquement synchronisé avec S3 avant exécution des modèles sur AWS.

**2️⃣ Utiliser SageMaker pour l’entraînement**

Dans **CDK**, tu peux créer un **SageMaker Training Job** qui prend les données depuis S3 et stocke le modèle dans S3 ou SageMaker Model Registry.

📌 **Exemple de définition CDK en Python (cdk\_sagemaker\_stack.py)**

python

CopierModifier

from aws\_cdk import (

core,

aws\_sagemaker as sagemaker,

)

class SageMakerStack(core.Stack):

def \_\_init\_\_(self, scope: core.Construct, id: str, \*\*kwargs):

super().\_\_init\_\_(scope, id, \*\*kwargs)

sagemaker.CfnTrainingJob(

self, "MLTrainingJob",

training\_job\_name="text-image-classification",

algorithm\_specification={

"trainingImage": "811284229777.dkr.ecr.us-east-1.amazonaws.com/xgboost:latest",

"trainingInputMode": "File"

},

role\_arn="arn:aws:iam::123456789012:role/SageMakerRole",

input\_data\_config=[{

"channelName": "train",

"dataSource": {

"s3DataSource": {

"s3Uri": "s3://my-bucket/processed/",

"s3DataType": "S3Prefix",

"s3DataDistributionType": "FullyReplicated"

}

},

"contentType": "csv"

}],

output\_data\_config={

"s3OutputPath": "s3://my-bucket/models/"

},

resource\_config={

"instanceType": "ml.m5.large",

"instanceCount": 1,

"volumeSizeInGB": 10

},

stopping\_condition={

"maxRuntimeInSeconds": 3600

}

)

✅ **Pourquoi c'est utile ?**  
✔ **Automatise l'entraînement du modèle** dans SageMaker avec les données de S3.  
✔ **Stocke les modèles entraînés directement dans S3 ou SageMaker Model Registry.**

**3️⃣ Automatiser l'inférence avec SageMaker Endpoint**

Après l’entraînement, tu peux **déployer ton modèle comme API REST sur AWS SageMaker**.

📌 **Exemple de déploiement d’un modèle en CDK (cdk\_sagemaker\_endpoint.py)**

python

CopierModifier

sagemaker.CfnEndpointConfig(

self, "SageMakerEndpointConfig",

endpoint\_config\_name="text-image-classification-config",

production\_variants=[{

"modelName": "text-image-classification-model",

"variantName": "AllTraffic",

"instanceType": "ml.m5.large",

"initialInstanceCount": 1

}]

)

sagemaker.CfnEndpoint(

self, "SageMakerEndpoint",

endpoint\_name="text-image-classification-endpoint",

endpoint\_config\_name="text-image-classification-config"

)

✅ **Avantages :**  
✔ **Transforme ton modèle en une API REST sur AWS SageMaker** 📡  
✔ **Accès scalable avec des requêtes en temps réel via AWS Lambda ou API Gateway**

**4️⃣ Intégrer AWS Lambda et API Gateway pour servir les prédictions**

Si tu veux **exposer ton modèle en tant qu'API publique**, tu peux le faire avec **AWS Lambda** et **API Gateway**.

📌 **Déploiement d’un modèle sur Lambda**

python

CopierModifier

import boto3

import json

sagemaker = boto3.client('sagemaker-runtime')

def lambda\_handler(event, context):

payload = json.loads(event["body"])

response = sagemaker.invoke\_endpoint(

EndpointName="text-image-classification-endpoint",

ContentType="application/json",

Body=json.dumps(payload)

)

result = json.loads(response['Body'].read().decode())

return {

"statusCode": 200,

"body": json.dumps({"prediction": result})

}

✅ **Pourquoi c’est utile ?**  
✔ **Permet d’envoyer des requêtes à l’endpoint SageMaker depuis une API REST.**  
✔ **Pas besoin d’une infrastructure lourde, tout est serverless avec Lambda.**

**🎯 Conclusion**

✅ **Oui, ta structure actuelle est bien adaptée pour un déploiement sur AWS.**  
🔹 **Quelques optimisations :**  
1️⃣ **Stocker data/ sur S3 et ajouter un script upload\_to\_s3.py**  
2️⃣ **Utiliser AWS SageMaker pour l'entraînement et l'inférence des modèles**  
3️⃣ **Déployer une API avec AWS Lambda et API Gateway pour exposer le modèle**  
4️⃣ **Créer une stack AWS CDK pour tout automatiser** 🚀

💡 **Tu veux un script Python pour uploader toutes tes données sur S3 automatiquement ?** 😊