**Explication de chaque fichier & architecture du projet :**

**Répertoire data (jeux de données)**

* **test\_set.csv :** fichier de test donné par le concours Kaggle (sur lequel notre modèle sera évalué)
* **test\_set\_cleaned\_addedFeatures.csv :** fichier de test donné par le concours Kaggle où on a rajouté les infos des hôtels à (à l’aide du fichier features\_hotels.csv) et les features extérieures (pib, nb touristes…).
* **features\_hotels.csv :** informations supplémentaires sur les hotels (hotel\_id, group, brand, city, parking, pool, children\_policy)
* **pricing\_requests\_done.csv :** toutes les requêtes effectuées depuis le début
* **all\_our\_requests\_done.csv :** cols: avatar\_id,mobile,language,city,date,id\_request (a quoi ça sert?)
* **soumission.csv : fichier créé par le fichier generation\_submission.py. Contient 2 colonnes : index, price.**

**Répertoire utilities (toutes les fonctions en .py)**

* **set\_path.py :** Fichier contenant les variables globales :
  + PATH : chemin absolu du dossier IA\_Framewotks\_Defi-IA-bedbugs 🡪 à modifier selon chaque ordinateur.
  + PATH\_CODE, PATH\_DATA, PATH\_MODELS, PATH\_ENCODING : chemin vers les repertoires code, data, models, encoding.
  + **NUM\_VAR, CAT\_VAR, INT\_VAR, DATES, CAT\_VAR\_FREQ : variables globales.**
* **Ce fichier est utile dans tous les .py qui vont suivre et devra donc être importé dans tous les .py**
* **data\_preprocessing.py : Regroupe les fonctions permettant d’effectuer le pré-traitement des données.**
* **models.py :** contient les différents modèles implémentés (XGboost, RanfomForest, Régression Linéaire Lasso, CATBoost)
* **generation\_submission.py :**  permet de générer le fichier « *soumission.csv* » pour effectuer une soumission Kaggle.
* **adversarial\_validation.py : Fichier permettant de faire l’adversarial validation càd de sélectionner les données du jeu d'entraînement (requêtes) les plus pertinentes i.e. celles qui s'approchent le plus de la distribution du jeu de test Kaggle.**
* **gradio\_defiIA.py :** met en place le Gradio.
* **Dockerfile : fichier Docker**

**Ancienne explications des fichiers (avant le rendu final)**

*J’avais du réaliser ce word pour mettre en forme tous les fichiers. Je laisse cette information à titre indicatif au cas où je souhaiterais reprendre le projet. Le projet avait été réalisé dans un groupe de 3 et ça a été très difficile car mes 2 camarades ne commentaient aucune fonction et n’avaient aucune structure dans leur travail. J’ai fait au mieux pour essayer de tout mettre au propre mais je n’ai jamais eu le temps de comprendre tous les codes que mes camarades ont fait.*

**Description des fichiers .csv :**

* **X\_train\_encoding.csv** : jeu d’entraînement avec target encoding des variables qualitatives
* **X\_val\_encoding.csv** : jeu de validation avec target encoding des variables qualitatives
* **X\_test\_encoding.csv** : jeu de test avec target encoding des variables qualitatives

------------------

* **Y\_train.csv** : labels du jeu d’entraînement (prix hotels)
* **Y\_val.csv** : labels du jeu de validation (prix hotels)
* **Y\_test.csv** : labels du jeu de test (prix hotels)

------------------

* **pricing\_requests\_done.csv** : toutes les requêtes effectuées depuis le début
* **pricing\_requests\_clean.csv :** toutes les requêtes effectuées depuis le début, sans les doublons
* **pricing\_requests\_clean\_addedFeatures.csv :** toutes les requêtes effectuées depuis le début, sans les doublons, avec ajout des features extérieures (pib, nb de touristes…)
* **features\_hotels.csv**: infos supplémentaires sur les hotels (hotel\_id,group,brand,city,parking,pool,children\_policy)

------------------

* **test\_set.csv** : fichier de test Kaggle (sur lequel notre modèle sera évalué)
* **test\_set\_cleaned\_addedFeatures.csv:** fichier de test Kaggle où on a rajouté les les infos des hotels de features\_hotels.csv et les features extérieures (pib, nb touristes…) (question: clean de quoi?)

------------------

* **tab\_id\_name\_of\_users.csv** : 2 cols : avatar\_id, avatar\_Name: tableau de correspondance entre l’avatar id et le nom de l’avatar comprenant la date de création de l’avatar.
* **allrequest.csv** : 3 cols: mobile, language, city: tableau de toutes les combinaisons possible des 3 variables (mobile, language, city) afin de piocher dans ce tableau les requêtes à faire.
* **all\_our\_requests\_done.csv:** cols: avatar\_id,mobile,language,city,date,id\_request (a quoi ça sert?)

------------------

* **submission.csv**: 2 cols: index, price (index de l’hotel? Ou index correspondant aux index de test\_set.csv?)
* **sample\_submission.csv**: 2 cols: index, price (index? difference avec submission.csv ?)

**Description des notebooks :**

**/.\ Je n’ai pas mis çà jour le git avec certains fichiers que j’ai ajouté/modifié.**

* **add\_features.ipynb** : fichier ajoutant de nouvelles features (à supprimer car il est inclus dans data\_preprocessing)
* **data\_drop\_duplicate.ipynb** : fichier qui
  + Élimine les données requêtées en double.
  + Crée une feature « *request\_nb »* qui indique si la requête a été faite en 1 en 2 ou en 3... par l'utilisateur.
  + Crée un fichier .csv : *pricing\_requests\_clean.csv* contenant les données nettoyées de doublons avec la nouvelle feature « *request\_nb »*
* **data\_preprocessing.ipynb**: fichier qui
  + Récupère « *pricing\_requests\_clean.csv »* et *« features\_hotels.csv »* et met en forme les données.
  + Effectue de la data visualisation sur « *pricing\_requests\_clean.csv »*

(à déplacer quand on mettra sous forme de fichier .py)

* + Ajoute des features extérieures comme : le PIB du pays, le prix moyen par m2 dans chaque ville, le nombre de touristes par an dans chaque ville, le nombre d’habitants par km2 dans chaque ville.
  + Renormalise les variables quantitatives « *price* » et « *stock* » pour les rendre plus gaussiennes.
  + Sépare le jeu de données en jeu de train, validation et test.
  + Performe de la sélection de features pour déterminer les features les plus importantes (à déplacer)
  + Encode ou non les variables qualitatives en dummy variables ou en target encoding et frequency encoding.
  + Crée les fichiers : *X\_train\_encoding.csv*, *X\_val\_encoding.csv*, *X\_test\_encoding.csv*, *Y\_train.csv*, *Y\_test.csv, Y\_val.csv*
* **ExploreTestSet.ipynb** : fichier faisant une data visualisation du jeu de test Kaggle afin de déterminer une première stratégie de requêtage avant même d’avoir implémenté des modèles. Fichier à conserver au format .ipynb
* **Generation\_soumission\_target\_encoding.ipynb**: fichier permettant de générer le fichier « *soumission.csv* » pour effectuer une soumission Kaggle. Ce fichier
  + Récupère le jeu de test Kaggle : *« test\_set.csv* » est le jeu de test Kaggle brut et « *test\_set\_cleaned\_addedFeatures.csv* » est le jeu de test Kaggle après ajout des features extérieures (pib, nb habitants…) (pourquoi clean ? Peut-être judicieux de le renommer test\_set\_kaggle)
  + Effectue le target encoding et frequency encoding des variables qualitatives.
  + Récupère le modèle
  + Applique le modèle au jeu de test Kaggle
  + Enregistre les résultats dans le fichier « *soumission.csv* »
* **Generation\_soumission.ipynb**: fichier permettant de générer le fichier « *soumission.csv* » pour effectuer une soumission Kaggle. Il s’agit du même fichier que *Generation\_soumission\_target\_encoding.ipynb*sauf que les variables qualitatives ne sont pas target encodées et frenquency encodées mais transformées en dummy\_variables.
* **pd\_profiling.ipynb** : fichier pour faire des profils report avec pandas. Utile ?
  + charge les fichiers *features\_hotels.csv* et *pricing\_requests\_done.csv* pour en faire des pandas profiling report.
* **Modeles\_commentés.ipynb** : Ce fichier permet d’appliquer les modèles
  + Récupère les fichiers *X\_train\_encoding.csv, X\_val\_encoding, X\_test\_encoding, Y\_train, Y\_test, Y\_val*
  + Implémente algorithme XGboost
    - Sans tuning des paramètres
    - Avec tuning des paramètres
  + Implémente algorithme RF
  + Implémente régression linéaire LASSO
  + Implémente CATBoost
* **Modeles.ipynb** : concaténation de *Modeles\_commentés.ipynb*  et *data\_preprocessing.ipynb.* A supprimer
* **request.ipynb :** code pour faire les requêtes à l’API. Le lasser sous format .ipynb ?
* **One\_Hot\_Encoding.ipynb** :
  + Récupère le fichier *pricing\_requests\_clean\_addedFeatures.csv'*
  + Effectue le one-hot-encoding des variables qualitatives
  + Split en un jeu de test et un jeu de train
  + Normalise les données
  + Crée un modèle XGBosst et l’applique aux données
* A supprimer ? Ca sert à quoi ?
* **adversarial\_validation.ipynb** : Ce fichier permet de sélectionner les données du jeu d'entraînement (requêtes) les plus pertinentes i.e. celles qui s'approchent le plus de la distribution du jeu de test Kaggle.
  + Fait l’adversarial validation
  + Enregistre les données sélectionnées sous le nom : *pricing\_requests\_clean\_addedFeatures\_****adv****.csv*
* **formating\_test\_set.ipynb :** Met en forme le jeu de test pour ajouter les nouvelles features : features issues de *hotel\_features.csv* et les features extérieures (pib, nb de touristes…)
  + Crée le fichier *test\_set\_cleaned\_addedFeatures.csv*
* **gradio\_defiIA.py**: fait le Gradio. Le seul fichier déjà mis en forme.

**Feuille de Route pour la transformation des .ipynb en .py :**

On va séparer le code en plusieurs fonctions. Chaque fonction devrait avoir un entête de la forme :

**def** train\_CatBoost**(**features,cat\_cols,adversarial\_train,

adversarial\_test,plot\_ROC = **False):**

"""

Format data for CatBoostClassifier and train model CatBoostClassifier.

It plots the ROC curve if specified.

Input:

-----

- features (list): all explanatory variables (categorical and numerical).

- cat\_cols (list): categorical xplanatory variables.

- adversarial\_train (dataframe): train dataset for adversarial validation.

- adversarial\_test (dataframe): test dataset for adversarial validation.

- plot\_ROC (bool): if True, plots the ROC curve.

Output:

------

- model: trained CatBoostClassifier model.

- train\_data: adversarial\_train formatted for CatBoostClassifier.

- holdout\_data: holdout\_data formatted for CatBoostClassifier.

"""

Le premier paragraphe explique à quoi sert la fonction. Si ce n’est pas évident, on peut aussi décrire comment on s’en sert. Les 2e et 3e paragraphes listent tous les inputs et outputs de la fonction avec leur type (si possible) et leur description rapide.

**Remarques générales :**

1. Je vous donne des idées de fonctions mais vous pouvez en rajouter/enlever pour faire ce qui vous semble le + logique.
2. Dans les ipynb, on charge souvent des données en début du notebook, par ex : pricing\_requests = pd.read\_csv(PATH + '//pricing\_requests\_clean.csv')

A la place de faire ça, on fait appel à **set\_path.py** qui va nous donner les variables globales à tous les fichiers.

1. Les fonctions ci-dessous dépendent les unes des autres, donc il faudra peut-être attendre qu’un .py soit créé pour construire le prochain. Dès qu’une d’entre nous à finit de créer le .py, elle le dit aux autres.
2. Placer tous les **.py** dans le dossier **utilities**
3. Placer les éventuels notebooks dans le dossier **notebooks**
4. Placer les éventuelles données dans le dossier **data**
5. Placer les modèles dans le dossier **models**
6. Créer d’autres dossiers si nécessaire ou des dossiers dans des sous-dossiers si c’est + pratique. Ce que j’écris n’est qu’une supposition.

**Fonctions à modifier et supposition d’organisation des fichiers :**

* Créer **set\_path.py** : (Lila) Ce fichier va contenir des variables globales :
  + PATH : chemin absolu du dossier Defi-IA-bedbugs 🡪 à modifier selon chaque ordi
  + DATA\_PATH : emplacement du dossier contenant les données (à créer)
  + MODEL\_PATH : emplacement du dossier contenant les modèles enregistrés
  + TARGET\_FREQ\_PATH : emplacement des poids du target et frequency encoding.
  + cat\_list, quant\_list, int\_list : variable globales immuables
* **Ce fichier est utile dans tous les .py qui vont suivre et devra donc être importé dans tous les .py**
* **data\_drop\_duplicate.ipynb** : (Flavie)
  + Transformer ce notebook en une seule fonction intitulée **data\_drop\_duplicates**()

(je pense que 1 fonction suffit). Prend en arg *all\_our\_requests\_done.csv* et renvoie *pricing\_requests\_clean.csv*

A voir dans quel .py on le met

* **adversarial\_validation.ipynb** : (Lila)

Créer **adversarial\_validation.py** et mettre dedans les fonctions suivantes :

* + packages nécessaires + **set\_path.py**
  + récupérer arg cat\_list, quant\_list, DATA\_PATH
  + **create\_adversarial\_data()**
  + **train\_CatBoost ()**
  + **plot\_conf\_matrix()**
  + **splited\_adversarial\_validation():** adversarial validation sur plusieurs morceaux
  + **main()** : fonction principale qui :
    - récupère avec DATA\_PATH, pricing\_requests\_clean\_addedFeatures.csv et

test\_set\_cleaned\_addedFeatures.csv

* + - Apelle **create\_adversarial\_data()**
    - Apelle **splited\_adversarial\_validation()**
    - Créer les données *pricing\_requests\_clean\_addedFeatures\_****adv*** contenant les données sélectionnées et faire une option pour les enregistrer au format.csv
* **data\_preprocessing.ipynb**: (Lila)

Créer **data\_preprocessing.py** et mettre dedans les fonctions suivantes :

* + Charger les packages nécessaires
  + Charger les path nécessaires avec **set\_path.py**
  + Charger les variables globales quant\_list et cat\_list de **set\_path.py**.
  + **type\_data()** : Récupère « *pricing\_requests\_clean.csv »* concaténé avec *« features\_hotels.csv »* et assigne le bon type aux données. Renvoie la table concaténée avec le bon type nommée *pricing\_requests*.
  + **add\_external\_features()** : Ajoute des features extérieures comme : le PIB du pays, le prix moyen par m2 dans chaque ville, le nombre de touristes par an dans chaque ville, le nombre d’habitants par km2 dans chaque ville. Récupère *pricing\_requests*, le modifie puis renvoie *pricing\_requests* avec les features ajoutées.
  + **normalise\_quanti()**: Renormalise les variables quantitatives « *price* » et « *stock* » pour les rendre plus gaussiennes. Mettre en argument quelles variables on veut normaliser (price, stock ou les 2)
  + **build\_train\_test()** : Sépare le jeu de données en jeu de train, validation et test. Prend en argument cat\_list, quant\_list, pricing\_request et renvoie X\_train, X\_val, X\_test, Y\_train, Y\_val, Y\_test.
  + **target\_freq\_encoding()** : argument pour savoir si on encode toutes les variables parail ou non. Prend en arg X\_train, Y\_train, X\_test et renvoie X\_train\_encoding, X\_test\_encoding, X\_val\_encoding (ou pas, garder le même non ?). Mettre une option pour enregistrer ou non les poids du modèle et dans le dossier TARGET\_FREQ\_PATH.
  + **oneHot\_encoding() :** fait le one-hot encoding de X\_train, X\_ val, X\_test. Renvoie de X\_train, X\_ val, X\_test ou sous de nouveaux noms : de X\_train\_oh, X\_ val\_oh, X\_test\_oh.
  + **main()** : mettre en arg si on veut faire du target enconding ou onehot encoding + récup la liste des variables sélectionnées par le feature sélection si il y a + récup les données de adversarial validation ou non.
    - Récupérer pricing\_requests\_clean.csv et features\_hotels.csv avec DATA\_PATH ou récupérer les données avec le **main()** de **adversarial\_validation.py**
    - Appel de **type\_data()**
    - Appel de **add\_external\_features()**
    - Appel de **normalise\_quanti()**
    - Appel de **build\_train\_test()**
    - Appel de **target\_freq\_encoding()** ou **oneHot\_encoding()**
    - Renvoie : X\_train\_encoding, X\_val\_encoding, X\_test\_encoding, Y\_train, Y\_test Y\_val (ou garder le nom X\_train, X\_val, X\_test). Mettre une option

pour pouvoir les enregistrer au format .csv ?

* **Modeles\_commentés.ipynb** : (Léa)

Créer **modeles.py** et mettre dedans :

* + Les packages nécessaires
  + **data\_preprocessing.py**
  + **set\_path.py**. Récupérer les variables nécessaires dans **set\_path.py**
  + **train\_XGboost()** : Implémente algorithme XGboost. Mettre en argument si on veut tuner les paramètres ou pas + mettre en argument si on veut sauver ou non le modèle dans un dossier particulier. Voir si on rajoute en plus une fonction **predict\_XGBoost()** ou si on met tout dans **train\_XGboost().**
  + **train\_RF()** : Implémente algorithme RF : même chose que pour XGbosst
  + **train\_lasso()** : Implémente régression linéaire LASSO : même chose que pour XGboost. Pas obligé de l’implémenter si ça donne de mauvais résultats à voir…
  + **train\_CATBoost ():** Implémente CATBoost : même chose que pour XGboost
  + **main()** : fonction principale qui
    - Prend en argument le nom du modèle qu’on veut appliquer
    - Appelle le **main()** de **data\_preprocessing.py** 🡪 on obtient X\_train, X\_test, X\_val bien encodés (one-hot ou target)
    - Entraîne un modèle au choix
    - Enregistre les poids dans le dossier situé à MODEL\_PATH
* **feature\_selection.py** : (Lila) Performe de la sélection de features pour déterminer les features les plus importantes. Contient :
  + reduceDataset() :
  + rfe() : renvoie liste des variables à garder
  + categorizeY() :
  + Chi2() : renvoie liste des variables à garder
  + MI() : renvoie liste des variables à garder
  + **main()** :
    - prend en arg X\_train,Y\_train,X\_val,Y\_val
    - Appèle reduceDataset pour réduire les données
    - Apelle rfe, chi2, mi..
    - Renvoie listes des variables à garder pour chaque méthode

A partir de là, je suis un peu perdue dans les rôle et l’utilisation des fonctions.

Mais il faudrait créer qu’un seul.py pour **Generation\_soumission\_target\_encoding.ipynb**et **Generation\_soumission\_one-hot\_encoding.ipynb**et il faudrait qu’on puisse choisir si on fait du target ou du one-hot, mais on verra comment on fait pour cette étape quand on aura déjà implémenté tout le reste

**+ Je ne sais pas à quoi sert One\_Hot\_Encoding.ipynb ??**

* **Generation\_soumission\_target\_encoding.ipynb**et **Generation\_soumission\_one-hot\_encoding.ipynb**(Flavie ou Léa ?)

Créer **generate\_submission.py** et mettre dedans :

* + Packages nécessaires
  + DATA\_PATH, quant\_list, cat\_list de **set\_path.py**.
  + Faire appel à la fonction **normalise\_quanti()**(déjà implémentée) pour transformer la variable stock
  + Récupérer **type\_data()** pour assigner le bon type aux données (si jamais la fonction n’est pas adapté, refaire le typage des données)
  + Récupérer **oneHot\_encoding()** ou **target\_freq\_encoding()** pour faire le target encoding ou le one hot encoding. L’option one-hot ou target encoding sera en argument
  + Récupérer les poids du modèle avec MODEL\_PATH
  + Récupérer le standard scaler
  + Sortir une prédiction
  + main() : …