CAS PRATIQUE 1
RAPPORT E2

Prédiction de la valeur médiane d'un quartier de logements californiens

Marianne DEPIERRE

20/02/2023

Titre professionnel RNCP "Développeur en intelligence artificielle"

Table des matières

1. Introduction

Ce rapport décrit un projet de machine learning dont l'objectif principal est de valider les compétences pour le titre professionnel RNCP « Développeur en Intelligence Artificielle ». Dans ce but, un modèle et son application sont repris afin d'être améliorés. Ce projet utilisait le jeu de données « California Housing Prices » contenant des informations provenant du recensement californien de 1990. Les données concernent les logements appartenant à un quartier californien (ou « block » en anglais).

1.1. Contexte

La startup Predimmo fournit des services pour les entreprises du domaine de l'investissement immobilier. Il y a plusieurs années, les chargés de relation client de cette startup ont relevé une augmentation de la demande pour la Silicon Valley. Il devenait difficile de répondre rapidement à toutes les demandes d'expertise pour un investissement immobilier dans cette région du monde. L'entreprise souhaitant automatiser cette tâche, elle a chargée un prestataire en intelligence artificielle de leur fournir un modèle prédictif.

Aujourd'hui, les chargés de mission font de nouvelles constatations. Les prédictions du modèle semblent en effet de moins en moins cohérentes avec les valeurs du marché. De plus, l'interface est jugée sobre et aucun élément ne permet à un nouvel utilisateur de se représenter les valeurs possibles d'un quartier californien. Predimmo a donc chargé un prestataire, mon entreprise, d'améliorer ces aspects de l'application et du modèle.

1.2. Enjeu et objectifs

Un modèle prédictif et une application ont été développés précédemment dans le but d'estimer la valeur d'un quartier de logements en Californie. En renseignant diverses informations dans un formulaire, l'utilisateur obtient une estimation du prix. Une révision du modèle et une amélioration graphique sont souhaitées par le client. Ainsi plusieurs objectifs ont été établis avec le client :

- Evaluation des performances du modèle existant
- Amélioration des performances de l'ancien modèle
- Amélioration graphique de l'application

Pour répondre aux besoins du client, un environnement de développement a été défini comme suit.

Langage	Librairies python	Outils
Python	Application : streamlit	Versioning : Git et GitHub
	Modèle et traitement de données : scikit-learn, pickle	
	Utilitaires : os, pandas, numpy	
	Versioning : mlflow	
	Tests unitaires unittest	
	Environnement virtuel : virtualenv	

Le langage python, ses librairies et les outils mentionnés permettent de lire et évaluer les performances de l'ancien modèle, d'optimiser celles-ci tout en vérifiant la non régression du modèle. Ils permettent aussi l'amélioration de l'interface de l'application.

1.3.Organisation de travail

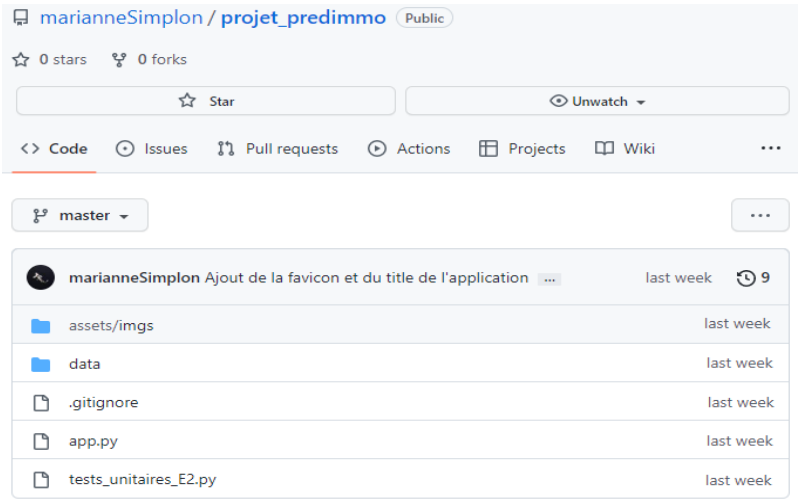
Dans un premier temps, un planning prévisionnel a été défini, montrant les dates et le temps alloué pour chacune des tâches identifiées.

Tâches	Description	26-déc	27-déc	28-déc	29-déc	30-déc
Définition du projet	Définition du besoin et de l'organisation du projet					
	Points avec le client					
Data Science	Reprise de l'existant du projet (notebooks, application et pickles)					
	Amélioration du modèle					
	Tests de non régression					
	Amélioration de l'application					
	Tests unitaires					
Livrables	Rédaction du rapport					
	Présentation orale					

Un planning réel a été effectué avec l'outil Trello. Il s'agit d'un outil de gestion de projet en ligne qui organise des projets avec des cartes représentant des tâches.

Globalement, le planning prévisionnel a été respecté. En effet, cinq jours étaient prévus pour la réalisation de ce projet et cinq jours y ont été consacrés. Les tâches ont été effectuées légèrement différemment. En effet, le rapport a été rédigé en parallèle de l'exécution des autres tâches, et l'amélioration du modèle et de l'application ont débuté le jour suivant. Des points avec le client ont été organisés et effectués en début et en fin de session de travail, pour le rendu des livrables.

Le versioning du projet s'est fait avec Git et GitHub.



2. Description de l'existant

2.1. Jeu de données

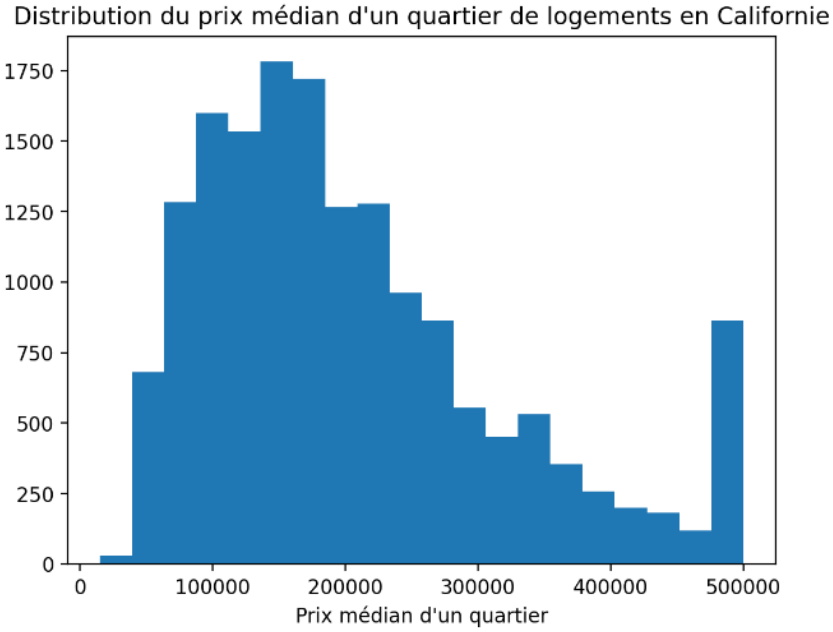
Le jeu de données est sous forme de fichier CSV.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-119.84	36.77	6.0	1853.0	473.0	1397.0	417.0	1.4817	72000.0	INLAND
1	-117.80	33.68	8.0	2032.0	349.0	862.0	340.0	6.9133	274100.0	<1H OCEAN
2	-120.19	36.60	25.0	875.0	214.0	931.0	214.0	1.5536	58300.0	INLAND
3	-118.32	34.10	31.0	622.0	229.0	597.0	227.0	1.5284	200000.0	<1H OCEAN
4	-121.23	37.79	21.0	1922.0	373.0	1130.0	372.0	4.0815	117900.0	INLAND

Il est composé de 16 336 lignes (ou observations) et 10 colonnes, soit une variable cible (ou target) et 9 variables explicatives (ou features). La tâche à réaliser est de type supervisée, plus précisément il s'agit d'un problème de régression linéaire car la variable cible est quantitative.

2.1.1. Variable cible ou « target »

La variable cible est la valeur médiane des logements pour les ménages d'un quartier, en dollars américain. Elle est nommée median_house_value dans le jeu de données. Il s'agit d'une variable numérique continue. Ci-dessous se trouvent un histogramme et des informations (pd.DataFrame.describe()) sur sa distribution.



Histogramme de la distribution de la variable cible

	median_house_value
count	16336.00
mean	206442.49
std	115264.34
min	14999.00
25%	119375.00
50%	179300.00
75%	264325.00
max	500001.00

Informations sur la distribution de la variable cible

2.1.2. Variables explicatives

Le tableau suivant récapitule les informations pour chacune des variables. Il résume aussi les informations accessibles grâces à la fonction pd.DataFrame.info() appliquées au dataframe du jeu de données.

Variable	Description	Type
longitude	Distance à l'ouest d'une maison.	float
latitude	Distance nord d'une maison.	float
house_median_age	Age médian d'une maison dans un quartier.	float
total_rooms	Nombre total de pièces dans un quartier.	float
total_bedrooms	Nombre total de chambres dans un quartier.	float
population	Nombre total de résidents dans un quartier.	float
households	Nombre total de ménages (ou groupe de personnes résidant dans un logement)	float
median_income	Revenus médian des ménages dans un quartier (en dizaines de milliers de dollars américains)	float
ocean_proximity	Distance à l'océan. Cinq catégories : « INLAND », « NEAR BAY », « NEAR OCEAN », « <1H OCEAN » et « ISLAND »	object

2.2. Application

L'application déjà en place utilise streamlit, une bibliothèque python qui permet de développer des applications à partir d'un script. En entrant des valeurs pour chacune des variables nécessaires à la prédiction, cette application estime le prix médian d'un quartier de logements en Californie (Annexe 1).

2.3. Modèle

Le modèle ainsi que le traitement des données existants sont récupérés sous formes de pickles. Les pickles en python sont utilisés pour sérialiser et dé-sérialiser une structure objet python. En résumé, un objet python est convertit en flux d'octets et est stocké dans un fichier, le pickle. Ce fichier contient toutes les informations pour reconstruire un objet python.

Le modèle existant est un modèle de « k-nearest neighbors » ou « KNN » de la bibliothèque Scikit-Learn. Les paramètres utilisés pour ce modèle sont ceux par défaut. Avant d'entraîner le modèles, les données ont été normalisées avec une fonction MinMaxScaler() et les données qualitatives ont été converties en valeurs numériques avec une fonction OneHotEncoder(). Les données manquantes ont été supprimées du jeu de données. Il n'y avait pas de doublons.

Dans un notebook Jupyter Notebook, les pickles sont exécutés sur le nouveau jeu de données et les scores du modèle existant sont récupérés.

Score	Valeur
R^2	0.6896264484170945
RMSE	63898.09826508343

- Le R^2 ou coefficient de détermination quantifie la corrélation dune ou des variables explicatives avec une variable cible. Il représente donc la proportion de la variation totale des données qui est expliquée par le modèle de régression. Ce coefficient peut être compris entre 0 et 1 et plus il est élevé, plus le modèle est considéré comme précis. Dans notre cas, il est égal à environ 0.69 ; cette valeur est un peu faible.

- L'Erreur Quadratique Moyenne ou « Root Mean Square Error » (RMSE) est une des méthodes mesurant la différence entre les valeurs prédites par le modèle et les valeurs réelles. Il s'agit de la métrique choisie pour ce problème de régression. Comme son nom l'indique, le RMSE correspond à la racine carrée de la moyenne des différences au carré entre une prédiction et sa valeur observée.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

Plus le RMSE est faible, meilleure est la performance du modèle. Le RMSE du modèle de base est d'environ 63 898. En d'autres termes, la racine carrée de la variance des résidus est égale à 63 898.
Si l'on compare à la moyenne (206 422 environ) et l'écart-type (115 264 environ) de la variable cible, le RMSE paraît trop important.

3. Améliorations

3.1. Amélioration du modèle

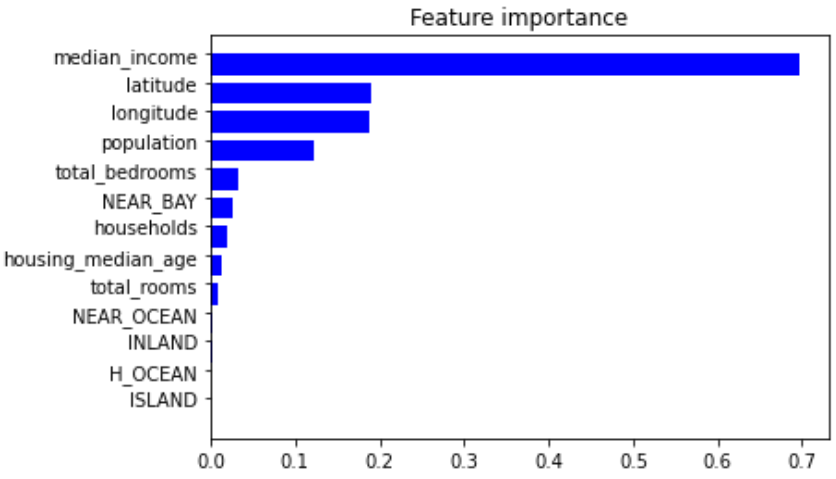
3.1.1. Feature selection (ou sélection des variables)

La feature selection (ou sélection de variables) consiste en la réduction du nombre des variables contribuant à un modèle prédictif. Elle montre plusieurs avantages :

- Les performances du modèle sont améliorées. En réduisant le nombre de variables, seules les plus importantes et les plus pertinentes sont conservées. Le risque d'overfitting (ou surajustement) est aussi réduit. En effet, en éliminant les variables donnant des informations redondantes, la sélection de variables prévient l'overfitting d'un modèle sur les données d'apprentissage et donc une mauvaise performance sur des nouvelles données.
- L'apprentissage du modèle est plus rapide et meilleur. Avec un nombre réduit de dimensions, les calculs de paramètres optimaux lors de l'apprentissage s'exécutent plus rapidement et plus efficacement.
- Le modèle est plus facile d'interprétation. Avec un ensemble plus restreint de variables, il est plus facile de comprendre et d'interpréter le modèle et ses résultats.

Précédemment, seule une colonne a été retirée de façon automatique dans le traitement des données. Il s'agit de la première colonne du one-hot encoding (ou encodage one-hot), c'est-à-dire « INLAND ».

Une feature selection a donc été réalisée en utilisant une méthode de feature importance (ou importance des variables). Cette technique assigne un score à chacune des variables en entrée basé sur leur capacité à prédire la variable cible. Dans ce processus, la fonction sklearn.inspection.permutation_importance() a été utilisée. Dans une feature importance par permutation, les valeurs de chaque variable explicative sont tour à tour mélangées aléatoirement. La diminution du score du modèle qui en résulte correspond à l'importance de cette variable. Ainsi, il n'y a plus de relation entre la variable explicative et la variable cible, et par conséquent la baisse de score indique à quel point le modèle dépend de cette variable explicative.



A l'issue de cette feature importance, les variables « ISLAND », « H_OCEAN », « INLAND », « NEAR_OCEAN », « housing_median_age », « total_rooms » ont été supprimées. En effet, celles-ci présentaient un score faible, inférieur à 0.02 environ. Au-delà, les performances du modèle diminuent.

A la suite de cette feature importance, la multicolinéarité est mesurée en calculant les Facteurs d'Inflation de la Variance ou « Variance Inflation Factors » (VIFs). La colinéarité est une association linéaire entre deux variables explicatives. Lorsqu'il y a une colinéarité, la corrélation est élevée et une partie de la variation d'une variable explique la variation de l'autre (l'une prédit l'autre). Il y a multicolinéarité lorsqu'au moins trois variables montrent de la colinéarité. Ainsi, des variables colinéaires sont problématiques puisqu'elles donnent des informations redondantes pour notre modèle.

	vif_index	features
4	88.138115	households
2	66.747902	total_bedrooms
3	13.898150	population
5	4.201990	median_income
0	4.185174	longitude
1	3.181850	latitude
6	2.039185	NEAR_BAY

Lorsque les VIFs sont inférieurs ou égaux à 1, il n'y a pas de variables montrant de la colinéarité. Il n'existe pas de consensus sur une valeur seuil qui démontrerait la multicolinéarité. Cette dernière est suspectée lorsque les valeurs sont supérieures à 1, mais communément au-dessus de 10 les variables sont analysées. Les variables « households », « total_bedrooms » et « population » sont concernées. En retirant « households » et « total_bedrooms », la variable « population » ne montre plus de colinéarité. Par conséquent seules les variables « households » et « total_bedrooms » sont supprimées.

En résumé, à l'issu de la feature selection les variables suivantes on été retirées : « ISLAND », « H_OCEAN », « INLAND », « NEAR_OCEAN », « housing_median_age », « total_rooms » « households » et « total_bedrooms ».

3.1.2. Sélection des variables explicatives ou « Fine Tuning »

Le fine-tuning (traduit par « réglage précis ») ajuste les hyperparamètres d'un modèle afin de mieux s'adapter aux données, et par conséquent améliorer ses performances. L'une des fonctions très utilisée pour réaliser un fine-tuning est sklearn.model_selection.GridSearchCV. Cette dernière recherche la meilleure combinaison d'hyperparamètres, c'est-à-dire celle qui maximise le score, à partir d'une grille. Cette grille correspond à des valeurs définies pour chacun des hyperparamètres. La fonction teste donc toutes les combinaisons possibles de la grille spécifiée.

Plusieurs grilles de paramètres ont été testées. Celle retenue est la suivante :

```
param_grid = [
    {
        'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
    }
]
```

Ainsi, seul l'hyperparamètre « n_neighbors », en d'autres termes, le nombre de voisins a été réglé. La valeur optimale pour ce paramètre est 9. Un nouvel entraînement du modèle a donc été exécuté avec « n_neighbors » égale à 9.

Les scores du nouveau modèle sont ceux affichés dans le tableau ci-dessous.

Score	Valeur
R^2	0.7421169669062511
RMSE	58266.46389388775

Au final, à l'issu de la feature selection et du fine-tuning, il y a une amélioration des scores, tant pour le RMSE (58 266), notre métrique choisie, que pour le R^2 (0.74).

3.2. Amélioration de l'application

Basiquement, l'application est très sobre et ne montre que les champs à remplir pour la prédiction. Conformément aux objectifs établis, l'application a également été améliorée (Annexe 1 et 2). Ces champs ont été mis à jour à l'issu de la révision du modèle. Une photographie de maisons de San Francisco a été insérée. Un menu déroulant a été intégré et permet, selon le désir de l'utilisateur, d'afficher en un clic la distribution de la variable à prédire. Enfin le logo de l'entreprise a été ajouté à l'application.

4. Vérifications de la non régression

4.1. Versioning avec MLflow

Du versioning a été réalisée avec la librairie mlflow. Le modèle est suivi pour vérifier qu'il n'y a pas de baisse de performance, notamment après la modification de ses hyperparamètres (Annexe 3). Le versioning avec mlflow ne montre aucune dégradation des performances du modèle.

4.2. Tests unitaires

Pour s'assurer que l'application n'avait pas régressée, les tests unitaires codés précédemment sont exécutés. Seuls les nombres de colonnes ont été modifiés, suite à la feature selection. Ces tests unitaires sont codés avec la librairie unittest et vérifient de la conformité :

- Du nombre de colonnes après collecte des entrées soumises par le formulaire
- Du type de la variable retournée après collecte des entrées soumises par le formulaire
- Du nombre de colonnes après traitement des données
- Du type de la variable retournée après traitement des données

Ci-dessous un extrait du code des tests unitaires réalisés.

```
class MyTestCase(unittest.TestCase):

    def test_inputs_to_df_type(self):
        self.assertEqual(
            type(inputs_to_df("-124.13", "40.80", 1259, "2.2478", "NEAR OCEAN")), pd.core.frame.DataFrame)
```

Après exécution du fichier (Annexe 4), les tests ont été passés avec succès.

```
(base) C:\Users\Admin\Documents\marianneSimplon\simplon\sentiment_analysis\projet_E2\app>python -m unittest tests_unitaires_E2.py -v
test_inputs_to_df_output_nbcol (tests_unitaires_E2.MyTestCase) ... ok
test_inputs_to_df_type (tests_unitaires_E2.MyTestCase) ... ok
test_preprocess_inputs_output_nbcol (tests_unitaires_E2.MyTestCase) ... ok
test_preprocess_inputs_type (tests_unitaires_E2.MyTestCase) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.070s

OK
```

4.3.Tests fonctionnels

Les tests fonctionnels de l'application ont été réalisés sous forme de jeu d'essais.

Cas d'essai	Résultat attendu	Résultat obtenu	Commentaires
Visibilité de l'image d'illustration	L'image s'affiche au chargement de la page, quelle que soit la taille de la fenêtre.	Conforme	
Affichage du menu déroulant	Le menu déroulant se déroule au clic de l'utilisateur	Conforme	
Affichage du graphique de la distribution de la variable cible	Le graphique est affiché lorsque le menu est déroulé.	Conforme	
Visibilité du formulaire	Le formulaire est visible quelle que soit la taille de la fenêtre	Conforme	
Affichage du formulaire	Le formulaire affiche les champs suivants : Longitude, Latitude, Age médian des logements, Nombre total de pièces, Nombre total de chambres, Nombre total de ménages, Revenu médian des ménages et Proximité de l'océan. Un bouton « Estimer le prix médian » permet de valider le formulaire.	Conforme	
Réussite de la prédiction	La prédiction est affichée en dessous du formulaire après soumission du formulaire	Conforme	
Echec de la prédiction	Un message d'erreur est affiché à l'utilisateur si la prédiction a échouée	Conforme	Prévoir un message personnalisé selon le type d'erreur améliorait l'expérience utilisateur.

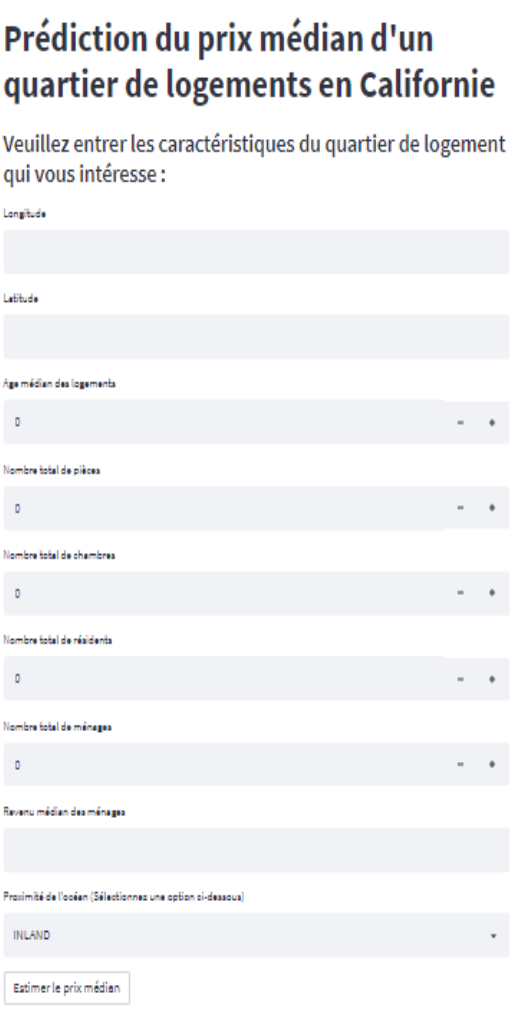
5. Bilan

Les besoins manifestés par le client ont été satisfaits. Les performances du modèle, notamment son RMSE, ont été améliorées grâce à une feature selection (permutation et VIF) et au fine-tuning (GridSearch sur n_neighbors). Ainsi le RMSE a été diminué d'environ 5 632, et le R^2 a été augmenté de 0.05 approximativement. L'apparence de l'application a été embellie d'une photographie et du logo de l'entreprise Predimmo. Un graphique de la distribution de la valeur médiane d'un quartier de logements californiens a été ajouté dans un menu déroulant à l'attention des nouveaux chargés de missions. Ceux–ci pourront se représenter dans quelle tranche de valeurs la prédiction se situe. L'application a été révisée afin de l'adapter au nouveau modèle et les tests de non régression ont été validés.

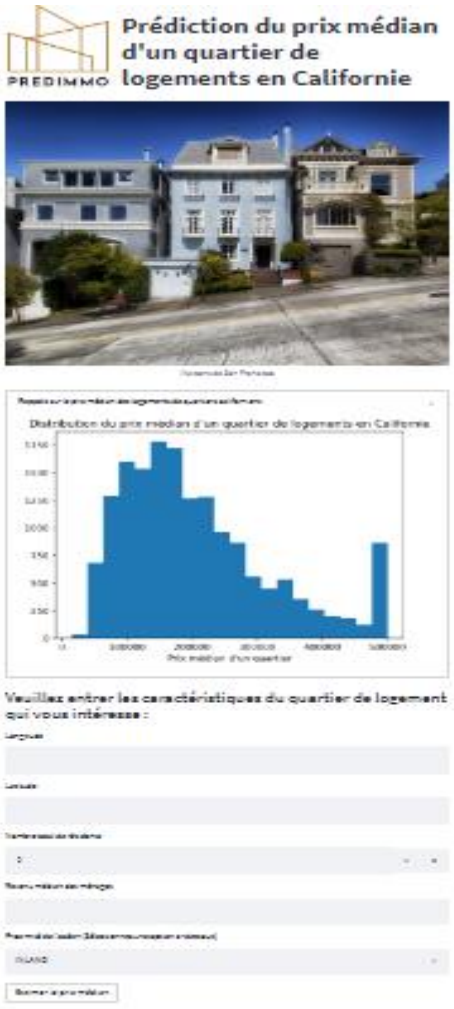
D'autres analyses permettraient potentiellement d'améliorer encore les performances du modèle, comme de la feature engineering (ou ingénierie des variables), mais n'ont pas été étudiées afin de tenir les délais du projet.

6. Annexes

Annexe 1 : Capture d'écran de l'application avant et après modification de l'interface



Capture de l'application existante



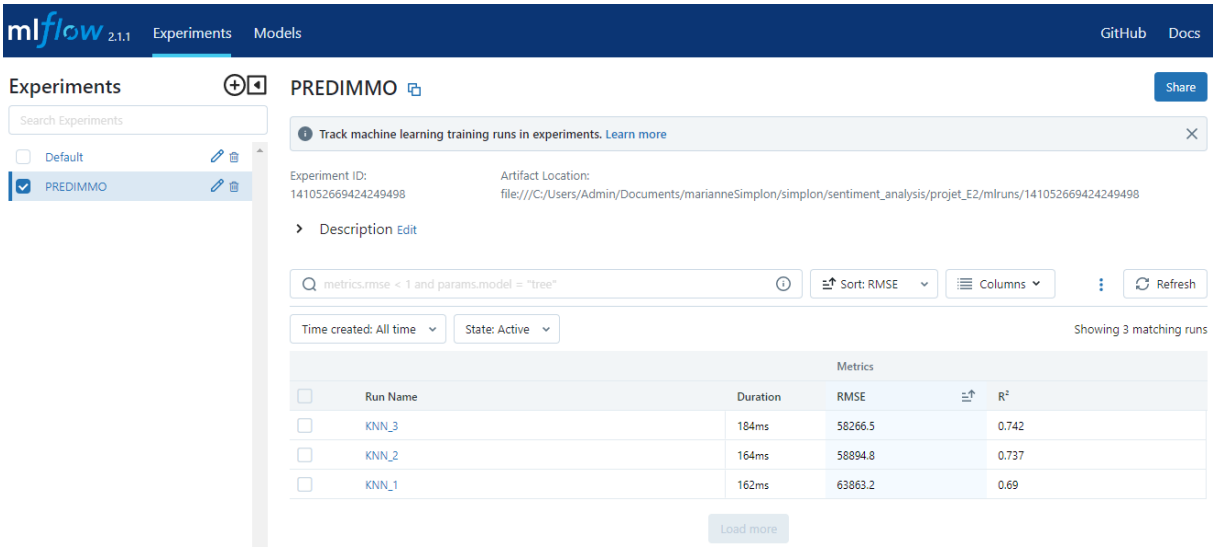
Capture de l'application après améliorations

Annexe 2 : Script python de l'application Streamlit

```
import os
import streamlit as st
import pandas as pd
import pickle
from PIL import Image
import matplotlib.pyplot as plt

### LOAD MODEL ###
MODEL_VERSION = 'knn.pkl'
MODEL_PATH = os.path.join(os.getcwd(), '../my_pickles',
                          MODEL_VERSION)  # path vers le modèle
with open(MODEL_PATH, 'rb') as handle:
    MODEL = pickle.load(handle)

### LOAD SCALER ###
SCALER_VERSION = 'minmaxscaler.pkl'
SCALER_PATH = os.path.join(os.getcwd(), '../my_pickles',
                           SCALER_VERSION)  # path vers le modèle
with open(SCALER_PATH, 'rb') as handle:
    SCALER = pickle.load(handle)

### LOAD ENCODER ###
ENCODER_VERSION = 'onehotencoder.pkl'
ENCODER_PATH = os.path.join(os.getcwd(), '../my_pickles',
                            ENCODER_VERSION)  # path vers le modèle
with open(ENCODER_PATH, 'rb') as handle:
    ENCODER = pickle.load(handle)

### INPUT RETRIEVAL ###
def inputs_to_df(longitude, latitude, population, median_income, ocean_proximity):
    data = {'longitude': [float(longitude)], 'latitude': [float(latitude)], 'population': [
        population], 'median_income': [float(median_income)], 'ocean_proximity': [ocean_proximity]}
    my_inputs_dataframe = pd.DataFrame(data)
    return my_inputs_dataframe


def preprocess_inputs(input_dataframe):
    non_cat = pd.DataFrame(input_dataframe[input_dataframe.columns[0:4]])
    cat = pd.DataFrame(input_dataframe[input_dataframe.columns[-1]])
    scale_data = pd.DataFrame(SCALER.transform(non_cat), columns=[
                              "longitude", "latitude", "population", "median_income"])
    encode_data = pd.DataFrame(pd.DataFrame.sparse.from_spmatrix(
        ENCODER.transform(cat)))
    encode_data.rename({0: 'H_OCEAN', 1: 'INLAND', 2: 'ISLAND',
                        3: 'NEAR_BAY', 4: 'NEAR_OCEAN'}, inplace=True, axis=1)
    encode_data = encode_data.drop(
        ["H_OCEAN", "INLAND", "ISLAND", "NEAR_OCEAN"], axis=1)
    preprocess_data = pd.concat([scale_data, encode_data], axis=1)
    return preprocess_data


def predict(preprocessed_input):
    pred = MODEL.predict(pd.DataFrame(preprocessed_input))
    return pred[0]
```

```
#########################################################################################################
########################################## STREAMLIT ####################################################
#########################################################################################################

### SETTINGS ###
st.set_page_config(
    page_title="PREDIMMO - Quartiers californiens",
    page_icon=Image.open('./assets/imgs/logo_predimmo.PNG')
)


### BODY ###

col1, col2 = st.columns([1, 3])
with col1:
    logo = Image.open('./assets/imgs/logo_predimmo.PNG')
    st.image(logo)

with col2:
    st.title("Prédiction du prix médian d'un quartier de logements en Californie")


image = Image.open('./assets/imgs/san-francisco-210230_960_720.jpg')
st.image(image, caption='Maisons de San Francisco')

### EXPANDER ###

with st.expander("Rappels sur le prix médian des logements de quartiers californiens"):
    df = pd.read_csv(
        r'./data/traindata_ori.csv', delimiter=',', decimal='.')
    median_house_value = df['median_house_value']
    fig, ax = plt.subplots()
    ax.hist(median_house_value, bins=20)
    ax.set_title(
        "Distribution du prix médian d'un quartier de logements en Californie")
    ax.set_xlabel("Prix médian d'un quartier")
    st.pyplot(fig)

### FORM ###

st.subheader(
    "Veuillez entrer les caractéristiques du quartier de logement qui vous intéresse :")

longitude = st.text_input('Longitude')

latitude = st.text_input('Latitude')

population = st.number_input('Nombre total de résidents', step=1)

median_income = st.text_input('Revenu médian des ménages')

ocean_proximity = st.selectbox(
    "Proximité de l'océan (Sélectionnez une option ci-dessous)",
    ('INLAND', '<1H OCEAN', 'NEAR BAY', 'NEAR OCEAN', 'ISLAND'))

if st.button(label="Estimer le prix médian"):
    try:
        datas = inputs_to_df(longitude, latitude, population,
                             median_income, ocean_proximity)
        if datas.isnull().values.any() == False:
            preprocess_datas = preprocess_inputs(datas)
            my_pred = predict(preprocess_datas)
            st.write('Prix médian estimé du quartier de logements :')
            st.write(my_pred)
        else:
            st.write(
                "Veuillez remplir tous les champs du formulaire s'il vous plait")
    except Exception:
        st.write("Une erreur s'est produite.")
```

Annexe 3 : Versioning du modèle avec mlflow



Annexe 4 : Script python réalisant les tests unitaires

```
import unittest
import pandas as pd
from app import inputs_to_df, preprocess_inputs, predict


class MyTestCase(unittest.TestCase):

    def test_inputs_to_df_type(self):
        self.assertEqual(
            type(inputs_to_df("-124.13", "40.80", 1259, "2.2478", "NEAR OCEAN")), pd.core.frame.DataFrame)

    def test_inputs_to_df_output_nbcol(self):
        self.assertEqual(
            len(inputs_to_df("-124.13", "40.80", 1259, "2.2478", "NEAR OCEAN").columns), 5)

    def test_preprocess_inputs_output_nbcol(self):
        self.assertEqual(len(preprocess_inputs(pd.DataFrame({'longitude': [float("-124.13")], 'latitude': [float(
            "40.80")], 'population': [1259], 'median_income': [float("2.2478")], 'ocean_proximity': ["NEAR
OCEAN"]})).columns), 5)

    def test_preprocess_inputs_type(self):
        self.assertEqual(
            type(preprocess_inputs(pd.DataFrame({'longitude': [float("-124.13")], 'latitude': [float("40.80")],
'population': [1259], 'median_income': [float("2.2478")], 'ocean_proximity': ["NEAR OCEAN"]}))),
pd.core.frame.DataFrame)


if __name__ == '__main__':
    unittest.main()
```