



TP Projet :

Rapport de fin de projet

Catil Gillian, Eyili Fatih, Le Devehat Mikael

La société Brew est une société de brassage de bière. Cette société cherche à développer et à innover afin de brasser une bonne bière. Cette société fait appel à l'intelligence artificiel afin d'aider dans le développement. Pour cela, l'intelligence artificiel doit prédire l'alcool par volume et l'amertume. Grâce à cette prédiction, la société pourrait accélérer le développement de ces nouvelles bières.

Pour commencer, Brew nous a donné une base de données en csv caractérisant une bière sans les données des ingrédients.

Les données fournies pour ce projet proviennent de Kaggle et permettent de caractériser 75 000 bières brassées avec 176 styles différents. Ces données ont divers attributs que nous devons utiliser afin de créer des prédictions sur l'amertume de la bière brassée ainsi que les totaux d'alcools de celle-ci. Pour cela, nous avons :

- BeerID : l'id correspond à la bière
- Name : le nom de la bière
- URL : le lien de la recette de bière
- Style : Type de la bière
- StyleID : Id du style de la bière
- Size(L) : Quantité brassée
- OG : Densité spécifique du moût avant fermentation
- FG : Densité spécifique du moût après fermentation
- ABV : alcool par volume
- IBU : L'unité d'amertume Internationale
- Color : référencement de la couleur de la bière
- BoilSize : le liquide au début de l'ébullition
- BoilTime : Le moment où le moût bouillit
- BoilGravity : Densité du moût avant ébullition
- Efficiency : Efficacité de l'extraction des sucres du grain pendant le moût
- MashThickness : quantité d'eau par livre de céréales
- SugarScale : Détermine la concentration de solides dissous dans le moût
- BrewMethod : La technique de brassage
- PitchRate : Levure ajoutée fermenteur par unité de gravité
- PrimaryTemp : Température du stade de fermentation

- PrimingMethod :
- PrimingAmount : Quantité de sucre de préparation utilisée
- UserId : l'id correspond à l'utilisateur

Ces données possèdent différentes valeurs dont des valeurs Null (ou NaN) ou des valeurs qui ne sont pas justes. Pour cela, nous devons nettoyer les données en fonction du nombre de valeur Null, du nombre de valeur aberrante, en fonction des colonnes utilisables. De plus, nous utilisons un système de profilage de Pandas (voir exemple : Annexe Profilage #1) pour connaître ce que doit nettoyer. Ainsi, les colonnes qui ne seront pas utilisées sont :

- BeerID :
- UserID
- URL
- NAME
- Style
- PrimingMethod
- PrimingAmount
- PitchRate
- MashThickness
- PrimaryTemp
- SugarScale
- BrewMethod

Les raisons sur le non-utilisation de ces colonnes sont dû au fait que :

- Nous n'avons pas besoin de l'id
- L'utilisation de string ne permet pas d'effectuer de calculs.
- Il y a trop de valeur Null (ou NaN)
- Il y a trop de valeur aberrante
- Des problèmes de corrélations
- L'utilisation de la colonne ne nous permet de créer des prédictions

Et nous nous assurons du nettoyage via des histogrammes (voir annexe Histogramme #1).

Maintenant, que les données sont nettoyées de tous types de problèmes, nous pouvons utiliser les données restantes afin de créer notre prédiction. Nous devons prédire le taux d'amertume (IBU) et d'alcool (AVB) d'une potentielle bière. Pour cela nous avons fait un tableau de corrélation.

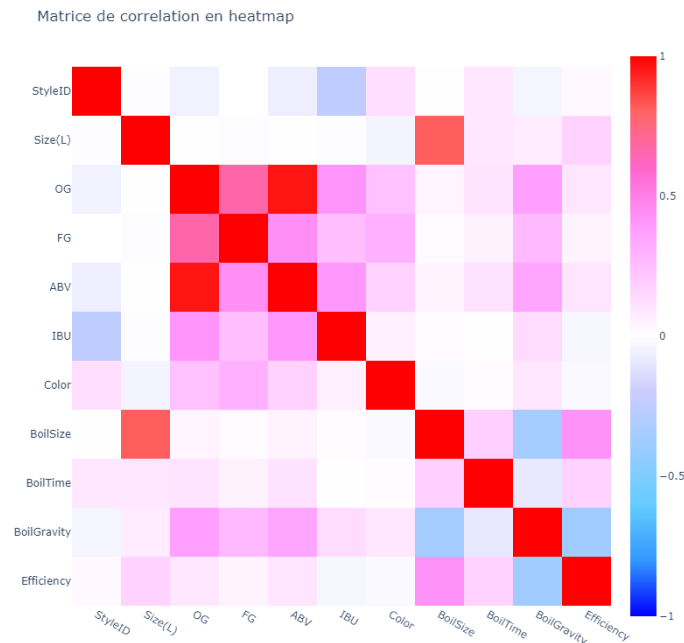
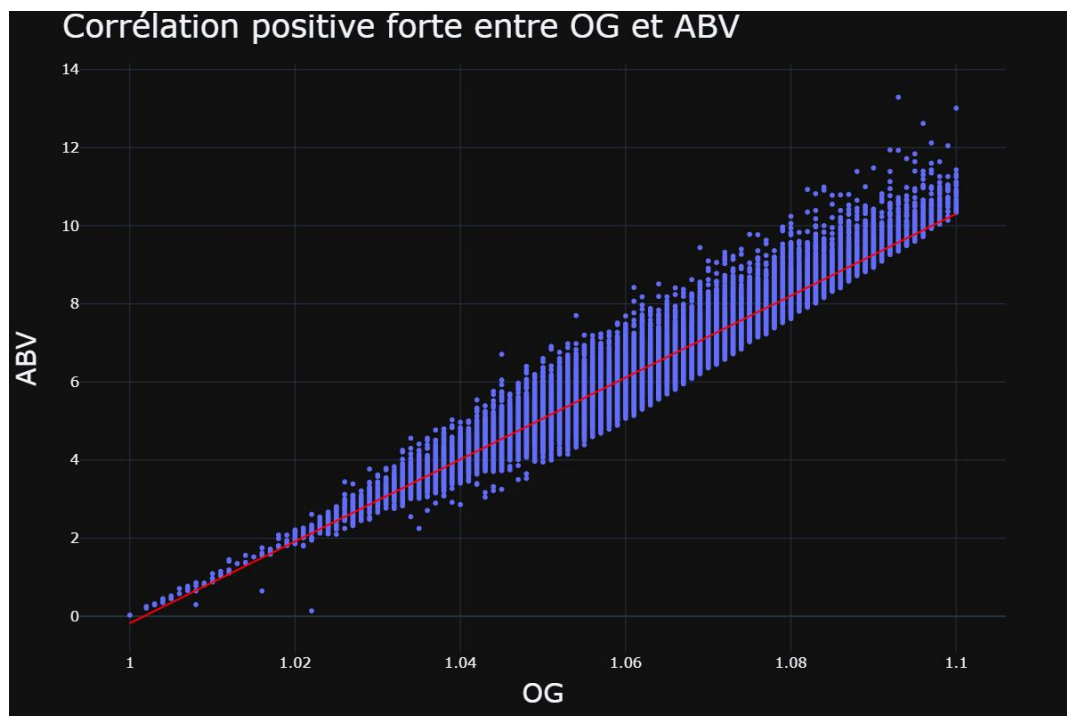


Diagramme de corrélation

Comme nous pouvons le voir via le tableau de corrélation, il y a une corrélation linéaire entre OG et ABV, ce qui nous permettra d'avoir une prédiction selon une courbe linéaire.



Pour cela, on sépare les données en 2 parties : une partie entraînement et une partie test. Pour cela on utilise le `train_test_split` venant de la bibliothèque `scikit-learn`. La partie test ne sera que 20% des données, les 80% restantes seront pour l'entraînement. Nous avons rajouté un `random_state` au sein de la séparation, pour que les données d'entraînement et de test soient séparées de manière aléatoire. Ensuite, on utilise le modèle de `LinearRegression` de `scikit-learn` sur lequel on entraîne les données d'entraînement.

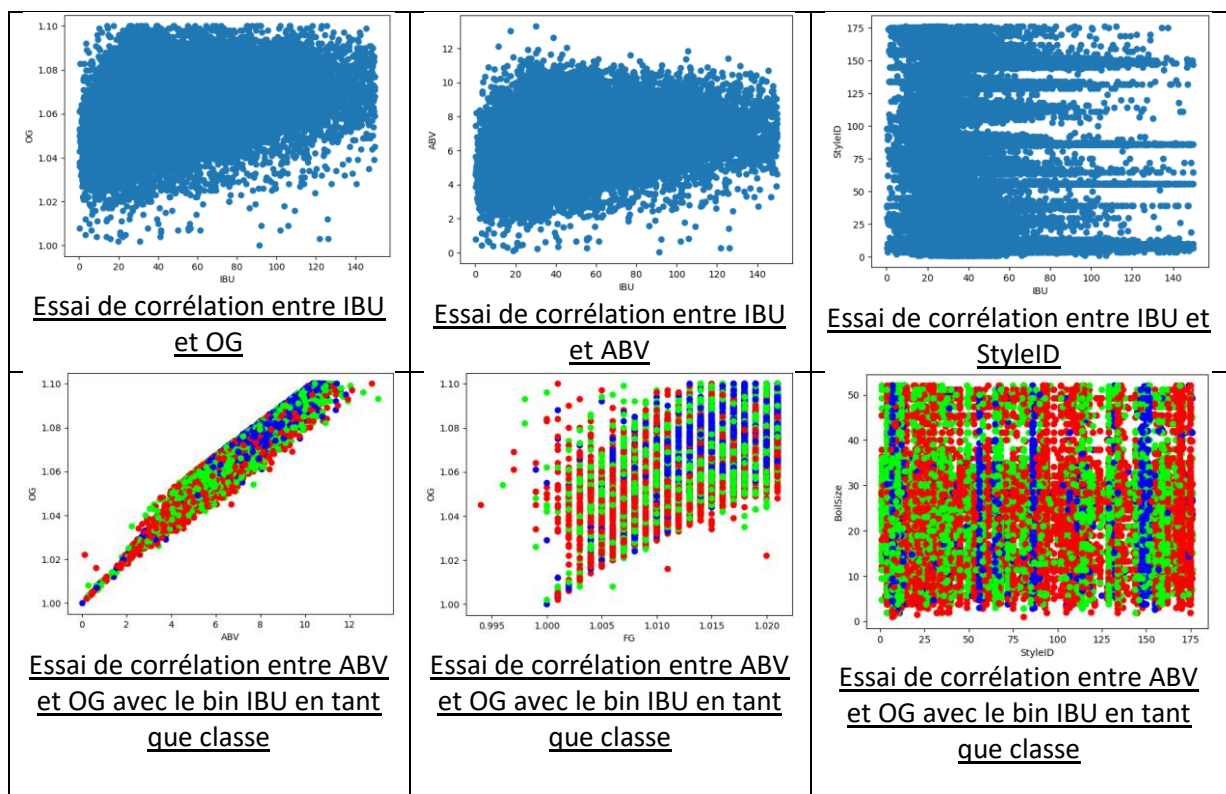
Enfin, nous pouvons tester les prédictions sur les données testes. Afin, de pouvoir les comparer aux données réelles, nous faisons une MSE (Mean Square Error). Plus la moyenne du carré de l'écart d'erreur est basse, plus la précision sur la prédiction est juste. Ici, le MSE est égal à :

$$\text{MSE} = 0.129432$$

La MAE (Mean Absolute Error) nous permet aussi de comparer les valeurs prédites aux valeurs réelles. Tout comme la MSE, plus elle est basse, meilleur est la précision. La MAE est égale à :

$$\text{MAE} = 0.281369$$

Pour IBU, La feature adéquat n'ai pas encore bien définit à travers le tableau de corrélation.



Exemple d'essai pour trouver une corrélation entre IBU et les autres données

Comme on peut le voir à travers ces essais graphiques, il est difficile de trouver un modèle de prédiction fiable avec un corrélation linéaire ou à 2 variables.

Ainsi, le choix d'utilisation de bin pourrait aider à prédire l'IBU via des catégories. Pour commencer, afin d'effectuer des tests nous avons défini les bins de tel sort que nous avons : [0,20), [20,40), [40,60), [60,80), [80,100), [100,150). Grâce à un one hot, nous avons les 6 classes que ce dessine via une représentation booléenne.

	[0, 20)	[20, 40)	[40, 60)	[60, 80)	[80, 100)	[100, 150)
0	True	False	False	False	False	False
1	False	False	False	True	False	False
5	False	False	True	False	False	False
7	True	False	False	False	False	False
8	False	True	False	False	False	False
...
73856	False	True	False	False	False	False
73857	False	False	True	False	False	False
73858	False	True	False	False	False	False
73859	False	True	False	False	False	False
73860	False	False	True	False	False	False

Les nouvelles données d'IBU, sera ensuite séparé en 2 parties : une partie entraînement et une partie teste. La partie entraînement servira à entraîner un RandomForestClassifier que l'on va ensuite tester via les données test. Il nous suffit de comparer la prédiction de celui avec les classes réelles de nos données ainsi nous arrivons à une précision de :

Accuracy: 0.21020148847340714

Malheureusement, les tests ont une précision de 21,02% de réussite. Pour continuer de tester les bins, nous sommes repartis avec les bins égales à [0,40) et [40,150), ainsi pour voir l'efficacité sur 2 classes au lieu 6 classes, et la précision monte jusqu'à 70%. Nous devons alors trouver un juste milieu.

Suite à quelques recherches et aux vues du nombre de données restantes suites au nettoyage, nous avons pris des bins égales aux intervalles : [0,30) renommé "Low", [30,60) renommé "Medium", [60,150) renommé "High". Grâce à cela, nous maximisons la précision tout en séparant les bières : en peu amère, moyennement amère et très amère. Puis, nous devons paramétrer efficacement le RandomForestClassifier afin d'améliorer la précision. Pour cela, nous utilisons HalvingGridSearchCV. HalvingGridSearchCV a besoin de connaître les paramètres possibles que l'on peut et que l'on varier. Dans le cas présent, nous avons créé un dictionnaire qui après plusieurs tests, ressemble à :

```
{"max_depth": [17, 18], "min_samples_leaf": [14, 15], "min_samples_split": [43, 44], "max_features": [None]}
```

Ce dictionnaire de possibilité de paramétrage sera lu par HalvingGridSearchCV, qui testera les divers paramétrages possibles pour ne garder le meilleur. Ensuite, nous le récupérons et nous l'entraînons, puis, nous le testons avec les données de la partie teste. Ainsi, l'efficacité évolue à :

Accuracy: 64.40

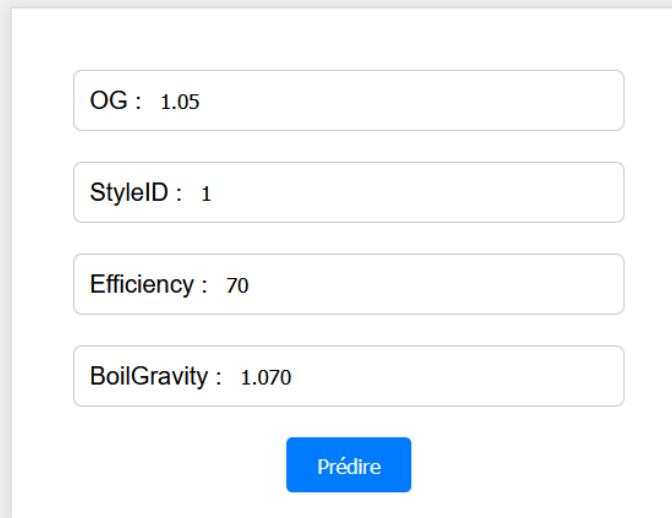
Ainsi, nous avons une précision de 64,4%. Malheureusement, nous gardons 35,6% d'erreur car les données, que nous avons, ne permettent pas de trouver précisément IBU correctement, ni le bin d'IBU.

Maintenant que nous avons les meilleures précisions possibles avec ces données pour prédire l'alcool par volume et l'unité d'amertume Internationale, nous devons avoir un affichage web pour inscrire les valeurs pour les nouvelles recettes de bière.

Pour cela, nous avons utilisé la technologie Flask. Ce Framework nous permet de réaliser une application web simple contenant un formulaire de saisie où seront inscrits les valeurs nécessaires pour la prédiction de ABV et IBU.

Projet IA - Mikael, Gillian, Fatih

Formulaire de prédiction pour ABV et IBU



OG : 1.05

StyleID : 1

Efficiency : 70

BoilGravity : 1.070

Prédire

Application web du programme

Tout d'abord, les valeurs entrées sont vérifiées pour savoir si se sont effectivement des nombres, sinon un message d'erreur est affiché, indiquant que la saisie est incorrecte.

Projet IA - Mikael, Gillian, Fatih

Formulaire de prédiction pour ABV et IBU

OG : m

StyleID : 1

Efficiency : 70

BoilGravity : 1.070

Prédire

Les informations entrées ne sont pas correctes!

Une fois les données entrées correctes, elles sont envoyées au server en POST par ajax, et sont traitées.

La valeur de OG est utilisée pour prédire AVB, puis StyleID, Efficiency, OG, BoilGravity et le ABV prédit sont stockés dans un Dataframe et utilisés pour prédire IBU. Enfin, si aucune erreur n'est survenue, un message JSON est renvoyé à l'application contenant les deux valeurs prédites avec un code 200 (succès), sinon un message d'erreur avec un code 400 est renvoyé. Enfin, ces deux valeurs sont affichées dans la page avec une légende correspondante.

Projet IA - Mikael, Gillian, Fatih

Formulaire de prédiction pour ABV et IBU

OG : 1.09

StyleID : 1

Efficiency : 70

BoilGravity : 1.070

Prédire

Voici quelles sont les prédictions :

ABV
9.2586

IBU
High

ABV: Représente le pourcentage d'alcool du volume de la bière

IBU: Représente l'amertume de la bière en PPM (parts par million), divisé en 3 catégories: Low(0-30), Medium(30-60), High(60-150)

Pour conclure, nous réussit à créer une IA qui, par le biais d'un nettoyage en prétraitement, de deux bons modèles et d'une application web, capable de prédire l'alcool par volume et l'amertume d'une bière. Bien que la précision de cette dernière soit pas proche de 100%, les résultats restent cohérents au vu de la non-corrélation entre IBU et le reste des données. Si nous voulions améliorer cette prédiction, il nous faudrait les données sur la composition de la bière car les ingrédients influencent l'amertume de cette boisson. Ainsi, cela évitera les erreurs de prédiction quand les réelles s'approchent des bornes des catégories d'amertume.

Annexes :

```
pd.set_option('display.max_columns', None) # a check c'est quoi

df = pd.read_csv('./recipeData.csv', encoding="latin1")

orig_leng = len(df)

df = df.drop(
    columns=["BeerID", "UserId", "URL", "Name", "Style", "PrimingMethod", "PrimingAmount", "PitchRate", "MashThickness",
            "PrimaryTemp", "SugarScale"]) # PrimaryTemp
df.dropna(inplace=True)
ser = df.isna().mean() * 100
|

# aze = df["Size(L)"].quantile(0.95)
df = df[df["Size(L)"] <= df["Size(L)"].quantile(0.95)]
df = df[df["OG"] <= df["OG"].quantile(0.95)]
df = df[df["FG"] <= df["FG"].quantile(0.95)]
df = df[(df["IBU"] <= 150) & (df["IBU"] > 0)] #IBU max == 150 selon wikipedia & supérieur a zero
df = df[df["BoilSize"] <= df["BoilSize"].quantile(0.95)]

hist = df.hist(bins=50, log=True)

new_len = len(df)
# print(new_len / orig_leng)

one_hot = pd.get_dummies(df["BrewMethod"])
print(one_hot)
df = df.drop(columns=["BrewMethod"])
df = df.join(one_hot)

print(df)
```

Code #1 : Code qui permet de nettoyer les données

```

from flask import Flask, request, jsonify, render_template
import re
import pandas as pd
from joblib import dump, load
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import train_test_split, HalvingGridSearchCV
from sklearn.metrics import accuracy_score, classification_report

app = Flask(__name__)
columns = ["OG", "StyleID", "Efficiency", "BoilGravity"]
columnIndexes = [2, 0, 10, 9]
Lr = load('../model_Linear_Regression.joblib')
Rf = load('../model_Random_Forest.joblib')
cat = ["Low", "Medium", "High"]

@app.route('/')
def index():
    return render_template('form.html', columns=columns)

@app.route('/query', methods=['POST'])
def traiter_requete_ajax():
    try:
        data = request.json

        dataForDf = {
            "OG": [float(data["OG"])],
            "StyleID": [int(round(float(data["StyleID"])))],
            "Efficiency": [float(data["Efficiency"])],
            "BoilGravity": [float(data["BoilGravity"])],
        }
        df = pd.DataFrame(dataForDf)
        abv = Lr.predict(df[["OG"]])
        df["ABV"] = abv

```

```

        ibu = Rf.predict(df[["StyleID", "Efficiency", "OG", "BoilGravity", "ABV"]])

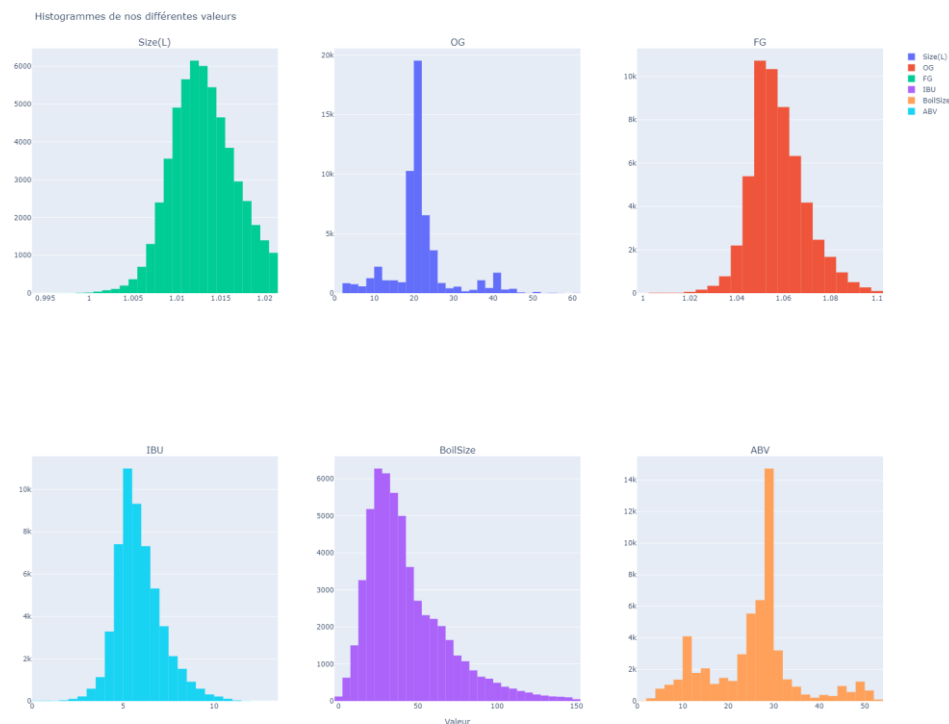
        abv = "{:.4f}".format(abv[0,0])

        for i in range(len(ibu[0])):
            if ibu[0,i] == True:
                ibu = cat[i]
                break

        return jsonify({"abv":abv,"ibu":ibu}), 200
    except Exception as e:
        return jsonify({'error': 'Erreur lors du traitement des données'}), 400

```

Code#2 du server python



Histogramme #1 : Vue d'ensemble des données



Profilage #1 : Gestion des Zéros pour IBU