# Compétition Kaggle

***Prédire si des clients vont retourner dans le restaurant dans les 90 jours***

## Compréhension des données

La première étape a été de comprendre et analyser les données. Pour cela rien de plus simple que de faire un .head() et .info() sur nos datasets pour avoir une vue d'ensemble.

Cela a permis de voir comment merger les différents datasets entre eux, grâce notamment à membe_id et restaurant_id. Une fois merger toutes les informations étaient réunis dans le même dataset.

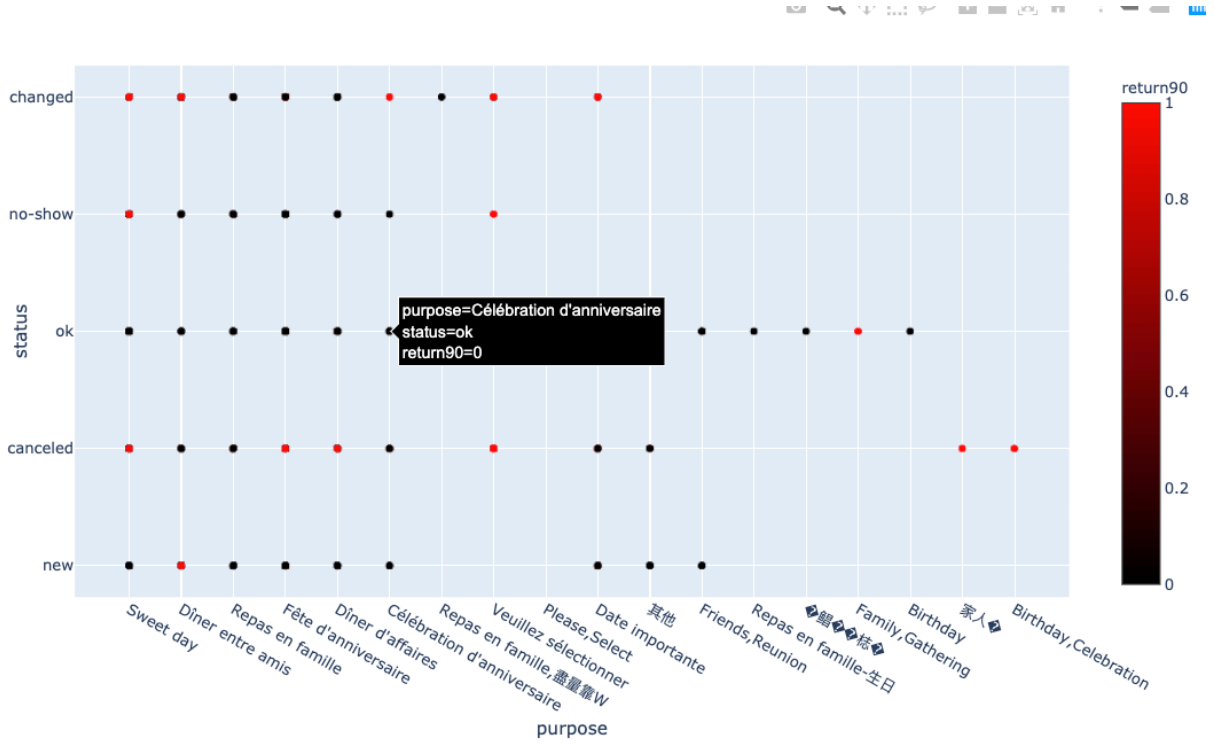
J'ai ensuite regardé la répartition des classes :

```
: data_booking[data_booking['return90']==1].count()
booking_id                      15613

: data_booking[data_booking['return90']==0].count()
booking_id                      62369
```

On voit tout de suite la répartition n'est pas uniforme entre les classes, on est de l'ordre du 75/25. J'ai gardé cela à l'esprit pour la suite, et notamment essayer d'utiliser des méthodes de re répartition de classe, cela n'a pas été convaincant.

J'ai ensuite plot les données, cela ne m'a pas apporté plus, mise à part de voir qu'il n'y avait pas vraiment de linéarité.




# Data & features engineering

Ensuite je me suis attaqué aux données en elles mêmes.

Beaucoup de données manquaient sur des milliers de lignes de chaque dataset. Supprimer les lignes avec de NaN aurait mené à une perte d'informations trop importantes. J'ai donc opté pour une méthode de remplacement des données manquantes.

Pour chaque columns si une données est manquantes elles remplacés par la valeurs la plus fréquente.

```
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

df_train_transformed = pd.DataFrame(imp.fit_transform(df_train))


df_train_transformed.columns = df_train.columns.values
```

Le résultats est une dataframe sans aucunes données manquantes.

J'ai ensuite drop plusieurs columns qui étaient inutiles ou en double suite au merge des datasets.

```
df_sub = df_sub.drop(['currency','name','abbr','tel','timezone','locale'],axis=1)
```

Viens ensuite la partie features engineering. La dataframe contient quelques dates il était donc primordiale d'en tirer des informations. 'cdate' et 'datetime' désignent respectivement la date de réservation et la date de visite du restaurant.

J'ai d'abord extrait pour chacun, le mois et l'année

```
df['visit_month'] = (pd.DatetimeIndex(df['datetime'])).month
```

```
df['resa_year'] = (pd.DatetimeIndex(df['cdate'])).year
```

J'ai ensuite voulu savoir si il s'agissait d'un jour de semaine ou pas, pour cela j'ai fais en sorte que les jour de 0 à 5 (exclu) prennent la valeur 1 et les autres la valeur 0.

```
df['weekday_resa'] = (pd.DatetimeIndex(df['cdate']).dayofweek).astype(float)
df['weekday_resa'] = (df['weekday_resa'] < 5).astype(float)
```

```
df['weekday_visit'] = (pd.DatetimeIndex(df['datetime']).dayofweek).astype(float)
df['weekday_visit'] = (df['weekday_visit'] < 5).astype(float)
```

J'ai ensuite plus au moins traité les saisons, grâce aux périodes de l'année, découpées en 4

```
df['quarter_resa'] = (pd.DatetimeIndex(df['cdate'])).quarter
```

```
df['quarter_visit'] = (pd.DatetimeIndex(df['datetime'])).quarter
```

Enfin, comme les champs de dates possèdent aussi l'heure, j'ai voulu extraire l'information du repas. Pour cela j'ai considéré différente tranches horaires. Avant 11h = le matin, de 11 à 16h = le midi et après 16h = le soir

```
: #(pd.DatetimeIndex(df['datetime'])).hour

  df['moment_repas'] = np.where((pd.DatetimeIndex(df['datetime'])).hour < 11, 'matin',
                                np.where((pd.DatetimeIndex(df['datetime'])).hour < 16 , 'midi', 'soir'))

  df['moment_repas']
```

```
0          midi
1         matin
2          midi
3         matin
4          midi
          ...
117061     soir
117062     midi
117063     midi
117064     midi
117065     midi
Name: moment_repas, Length: 117066, dtype: object
```

# Training des modèles

Pour commencer par simple la première approche a été une logistique regression. Contre toute attente les résultats ont été correct, au alentour de 0.63 en ROC_AUC score.

Avec un peu de features sélection et surtout la transformation des variables catégorielles en variables binaires grâce à la méthode .get_dummies() de pandas, le score du modèle s'est un peu amélioré pour atteindre aux alentours de 0.66.

Après plusieurs essais notamment avec du hyper parameter tuning de la logistique regression, impossible de dépasser 0.66.

Je suis alors passé à des méthodes d'ensembles, gradient boosting avec XGBoost.

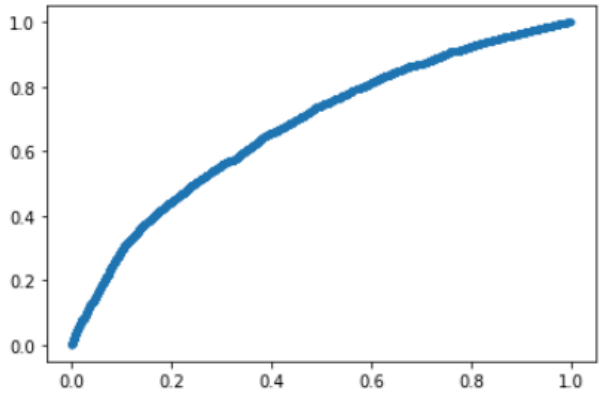
Avec de l'hyper parameter tuning et de la cross validation sur XGBoost j'ai pu dépassé 0.66 pour atteindre environ 0.676

```
            "silent": 1,
        }

params = {"learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
          "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
          "min_child_weight" : [ 1, 3, 5, 7 ],
          "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
          "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ] }
```

voici les paramètres utilisés sur le GridSearch pour trouver les paramètres optimaux

```
AUC: 0.6786141827625043
```




# CatBoost

Après avoir tune au max mon modèle XGBoost, je n'arrivais plus à augmenter le score, j'ai donc voulu essayer un modèle appelé CatBoost et que j'ai découvert en suivant un cours sur les méthodes d'ensembles sur Datacamp.

Il s'agit d'un algorithm de gradient boosting, basé sur des decision tree, open source.

Il a 2 avantage principaux :

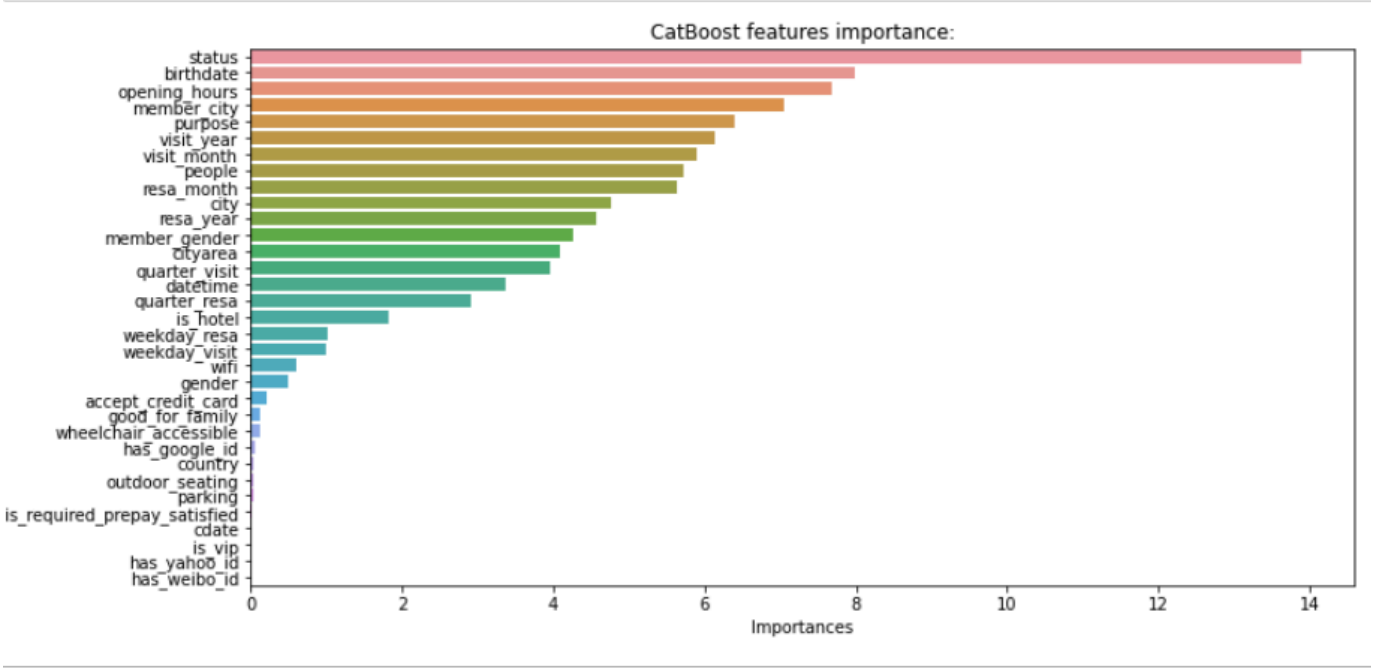
- il est en général plus rapide que XGBoost et LightGBM surtout sur le temps de prédiction.
- Il a un excellent support des variables catégorielles, pour cela il utilise le one-hot-encoding.

Étant donné que notre jeu de données a énormément de variables catégorielles, j'ai pensé qu'il serait intéressant d'essayer cette algorithme.

Les résultats ont été au rendez-vous et j'en ai fais mon modèle principal pour la compétition Kaggle.

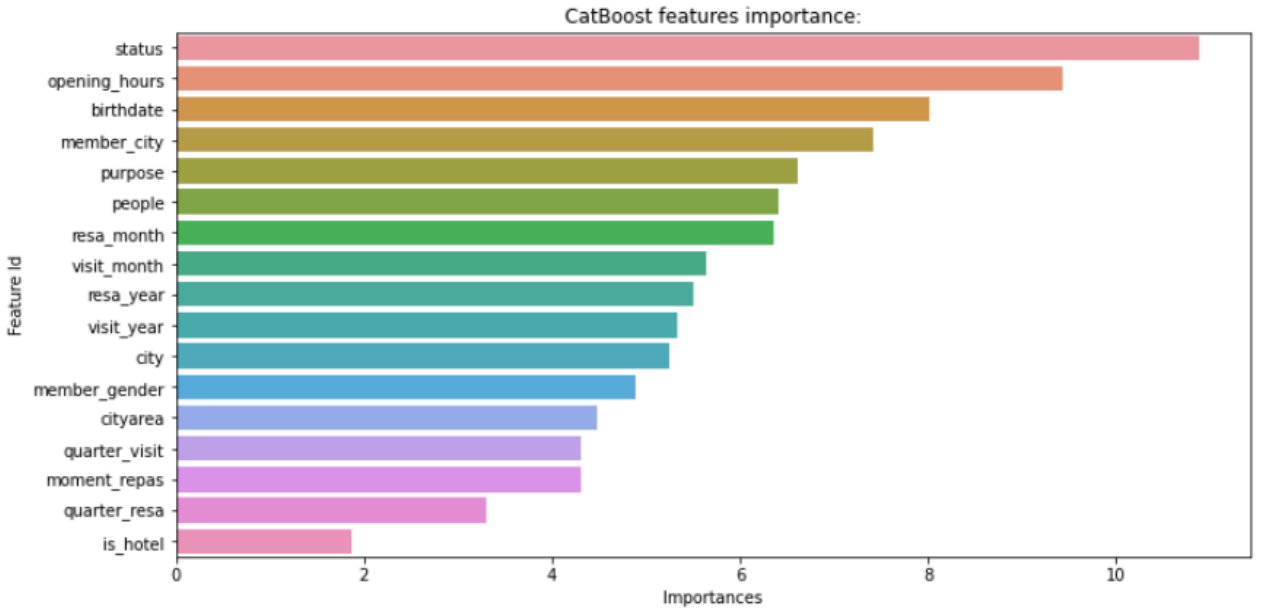
L'hyper parameter tuning a prit 1h30 avec CatBoost, là ou il en avait pris 4h avec XGBoost.

Après avoir tune le modèle j'ai voulu me tourner vers l'importance des variables sur le modèles.




Dans le but de rendre le modèles plus lisible et surtout de ne garder que les variables essentielles, j'en ai supprimé plusieurs.

Maintenant nous avons une vision beaucoup plus claire de l'importance de chaque variables sur le modèle. Sans grande surprise, le status à l'impact le plus fort.




La surprise se situe plus au niveau de 'birthdate', même si il s'agit d'une date je le l'ai pas traité car, les dates n'avaient pas toutes le mêmes format et plus d'un tier des 'birthdate' étaient nul. Catboost la traite en temps que variables catégorielles et réunis les dates similaires entre elles pour en faire des catégorie, étant donné que dans un dataset de 100 000 lignes il y a forcément beaucoup de personnes nées le même jours.