

# REMERCIEMENTS

Je souhaite adresser mes remerciements à DataScientest pour m'avoir permis d'accéder à une formation correspondant à mes attentes.

Je souhaite plus particulièrement remercier Lara et Dimitri pour m'avoir permis d'atteindre mes objectifs ainsi que l'accompagnement que j'ai eu lors de ce projet.



## I. INTRODUCTION

### a. Contexte

Lors d'une campagne de Marketing d'une banque, l'analyse des données est une problématique très classique dans les entreprises de service, pour ce faire, j'ai pris sur un jeu de données, des données personnelles sur des clients d'une banque qui ont été « télémarketés » pour souscrire à un produit que l'on appelle un « dépôt à terme ». Le but est lorsqu'un client souscrit à ce produit, il place une quantité d'argent dans un compte spécifique et ne pourra pas toucher ces fonds avant l'expiration du terme. En échange, le client reçoit des intérêts de la part de la banque à la fin du terme.

### b. Objectifs et enjeux

Trouvez les meilleures stratégies à améliorer pour la campagne marketing. Comment l'institution financière peut-elle avoir une plus grande efficacité pour les futures campagnes de marketing ?

Les enjeux sont d'effectuer une analyse visuelle et statistique des facteurs pouvant expliquer le lien entre les données personnelles du client (âge, statut marital, quantité d'argent placé dans la banque, nombre de fois que le client a été contacté, etc.) et la variable cible "Est-ce que le client a souscrit au dépôt à terme ?"

Ceci se fera en quatre parties :

- **Exploration de données** : data processing, data cleaning, visualisation des données, tests statistiques.
- **L'analyse visuelle** : utilisation des techniques de machine learning pour déterminer à l'avance si un client va souscrire au produit ou non.
- **Prédiction** : utilisation des techniques d'interprétabilité des modèles de machine learning pour expliquer à l'échelle d'un individu pourquoi il est plus susceptible de souscrire au produit ou non.
- **L'interprétabilité du modèle** : conclusion de mon analyse.

## II. EXPLORATION DES DONNÉES

### a. Présentation du DataSet

Pour mon étude, je dispose d'une source de données :

Le jeu provient du site Kaggle (<https://archive.ics.uci.edu/ml/datasets/bank+marketing>). Il s'agit d'un unique fichier csv qui recense différentes informations au sujet des clients qui ont été contactés lors des différentes campagnes de télémarketing.

### b. Variables

Je commence l'exploration du DataSet, qui sera nommé *df*.

Celui-ci comporte 45211 lignes et 17 variables. Représentées de la manière suivante :

Nom de la variable	Variable cible ou explicative	Description	Type	Variable catégorielle ou quantitative
Age	Explicative	Age du client	Int64	Catégorielle
Job	Explicative	Métier du client	Object	Catégorielle
Marital	Explicative	Situation familiale	Object	Catégorielle
Education	Explicative	Secteur d'activité du client	Object	Catégorielle
Default	Explicative	Solde du compte bancaire positif ou négatif	Object	Catégorielle
Balance	Explicative	Argent sur le compte du client	Int64	Quantitative
Housing	Explicative	Propriétaire d'une maison	Object	Catégorielle
Loan	Explicative	Crédit en cours	Object	Catégorielle
Contact	Explicative	Type de contact	Object	Catégorielle
Day	Explicative	Jour de contact	Int64	Catégorielle
Month	Explicative	Mois de contact	Object	Catégorielle
Duration	Explicative	Durée du dépôt	Int64	Quantitative
Campaign	Explicative	Numéro de la campagne d'appel	Int64	Catégorielle
Pdays	Explicative	Nombre de jour depuis le dernier contact client	Int64	Quantitative
Previous	Explicative	Nombre de contact du client	Int64	Quantitative
Poutcome	Explicative	Succès si contacté ou non	Object	Catégorielle
y	Cible	Souscription au produit de placement financier	Object	Catégorielle

Sur l'ensemble des 17 variables de *df*, il y a :

- 16 variables explicatives (12 catégorielles + 4 quantitatives). Elles correspondent aux informations des clients qui ont été contactés par l'une des campagnes d'appels téléphonique.
- 1 variable cible (catégorielle) « *y* ». Elle indique si le client a souscrit ou non au produit de placement financier.

Toutes les variables de *df* sont aisément compréhensibles, et il n'y a aucunes valeurs manquantes, ni de doublon dans le DataSet.

```
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   age         45211 non-null    int64  
 1   job          45211 non-null    object  
 2   marital      45211 non-null    object  
 3   education    45211 non-null    object  
 4   default      45211 non-null    object  
 5   balance      45211 non-null    int64  
 6   housing      45211 non-null    object  
 7   loan          45211 non-null    object  
 8   contact       45211 non-null    object  
 9   day           45211 non-null    int64  
 10  month         45211 non-null    object  
 11  duration      45211 non-null    int64  
 12  campaign      45211 non-null    int64  
 13  pdays         45211 non-null    int64  
 14  previous      45211 non-null    int64  
 15  poutcome      45211 non-null    object  
 16  y              45211 non-null    object  
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

c. Gestion des valeurs manquantes et des doublons

Je constate qu'il n'y a aucune valeur manquante, ni de doublons sur l'ensemble du DataSet :

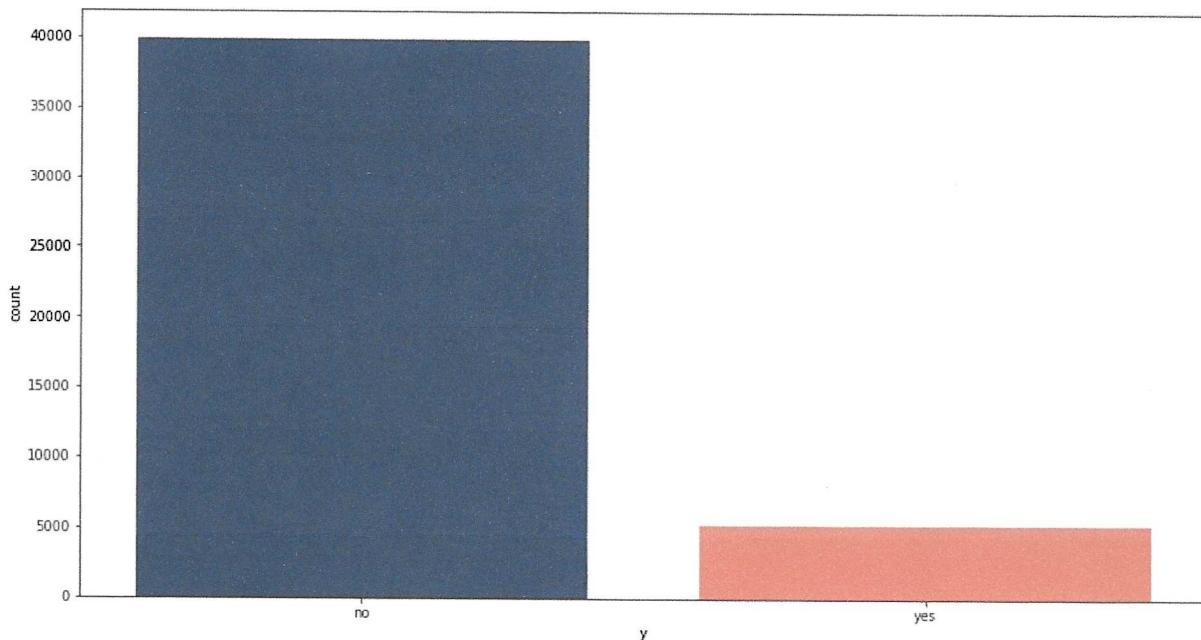
```
df.duplicated().sum()  
0  
  
# Est-ce qu'il y a des Nans ?  
df.isna().sum()  
  
age          0  
job          0  
marital      0  
education    0  
default      0  
balance      0  
housing      0  
loan          0  
contact      0  
day           0  
month         0  
duration     0  
campaign     0  
pdays        0  
previous     0  
poutcome     0  
y             0  
dtype: int64
```

### III. DATA VISUALISATION

#### a. Distribution de la variable cible « y »

Ma première idée de visualisation est d'observer la distribution de ma variable cible « y » :

```
plt.figure(figsize =(15,8))  
custom_palette = ["#001166", "#F95E4C"]  
sns.set_palette(custom_palette)  
ax = sns.countplot(x = "y", data = df);
```



Je remarque un déséquilibre sur ma variable « y », ceci est un élément à prendre en compte lors de la prédiction. Puisque le résultat « no » va avoir une influence importante sur celle-ci.

b. Distribution des variables par rapport à la variable cible « y »

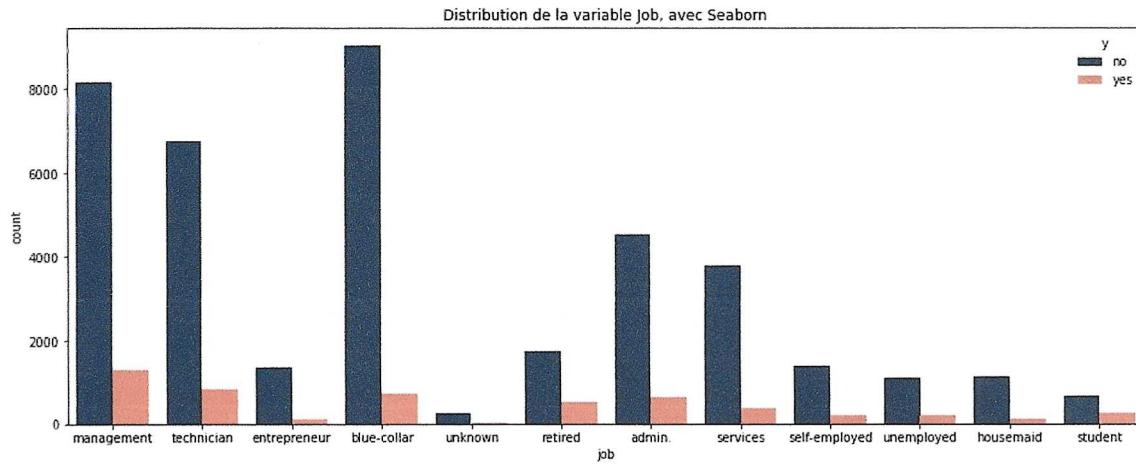
Pour continuer à voir la distribution des autres variables, je décide de mettre en contraste la variable cible avec les variables que je dispose.

Variable : « job » : indique le métier du client. Variable catégorielle.

Objectif : Analyser si le métier a une influence sur la souscription au produit de placement financier.

```
# Création de la figure
fig = plt.figure(figsize = (16, 6))

# Avec Seaborn en ajoutant les couleurs par rapport à la variable deposit
sns.set_palette(custom_palette)
sns.countplot(x = 'job', hue = 'y', data=df)
plt.title("Distribution de la variable Job, avec Seaborn")
```



Je remarque que les clients contactés sont principalement des managers, des techniciens, des personnes travaillant dans l'administratif, des métiers du bâtiment, des retraités.

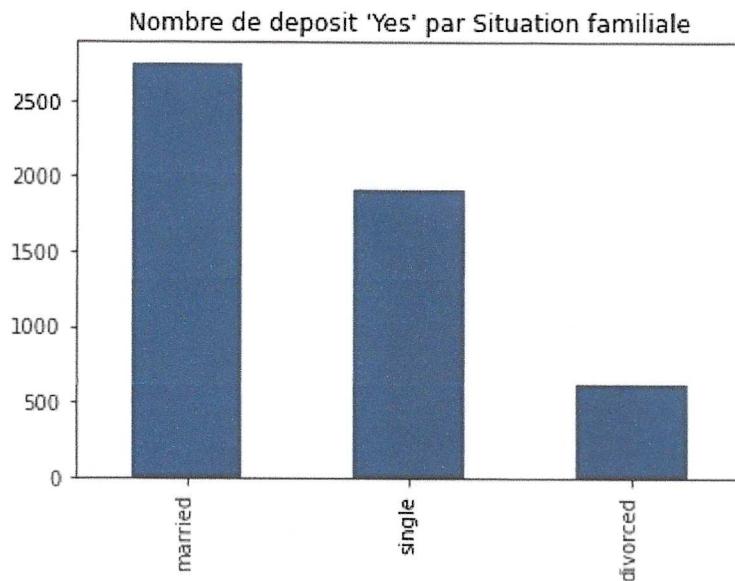
Cela peut venir du produit de placement proposé aux personnes. Ne connaissant pas les modalités du produit, il est compliqué de déterminer si le produit influe sur la typologie des personnes susceptibles d'être intéressées.

Variable : « marital » : indique la situation familiale. Variable catégorielle.

Objectif : Analyser si la situation familiale a une influence sur la souscription au produit de placement financier.

```
# Avec Pandas voir la distribution des deposit "Yes" par Situation familiale
# Creation d'un DataFrame "deposit" == "yes" a partir de la colonne "marital"
deposit_marital_yes = df[(df["y"] == "yes") & (df["marital"])]
```

```
# Creation du graphique
deposit_marital_yes['marital'].value_counts().plot.bar()
plt.title("Nombre de deposit 'Yes' par Situation familiale");
```



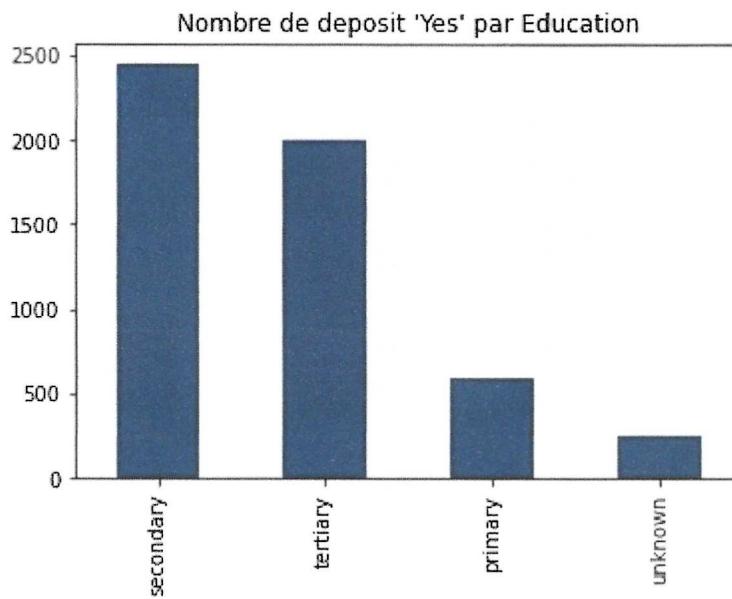
Je remarque que les personnes ayant souscrit au produit de placement financier, sont majoritairement des personnes mariées ou célibataires.

On peut supposer que les personnes se projetant dans l'avenir et voulant préparer un projet demandant des ressources financières sont plus intéressé par ce type de produit.

Variable : « education » : indique le secteur d'activité. Variable catégorielle.

Objectif : Analyser si le secteur d'activité a une influence sur la souscription au produit de placement financier.

```
# Avec Pandas voir la distribution des deposit "Yes" par Education
# Crédit à partir de la colonne "education"
deposit_education_yes = df[(df["y"] == "yes") & (df["education"])]  
  
# Crédit du graphique
deposit_education_yes['education'].value_counts().plot.bar()
plt.title("Nombre de deposit 'Yes' par Education");
```



Je remarque que les personnes ayant souscrit au produit de placement financier sont principalement des personnes évoluant dans le secteur d'activité secondaire et tertiaire.

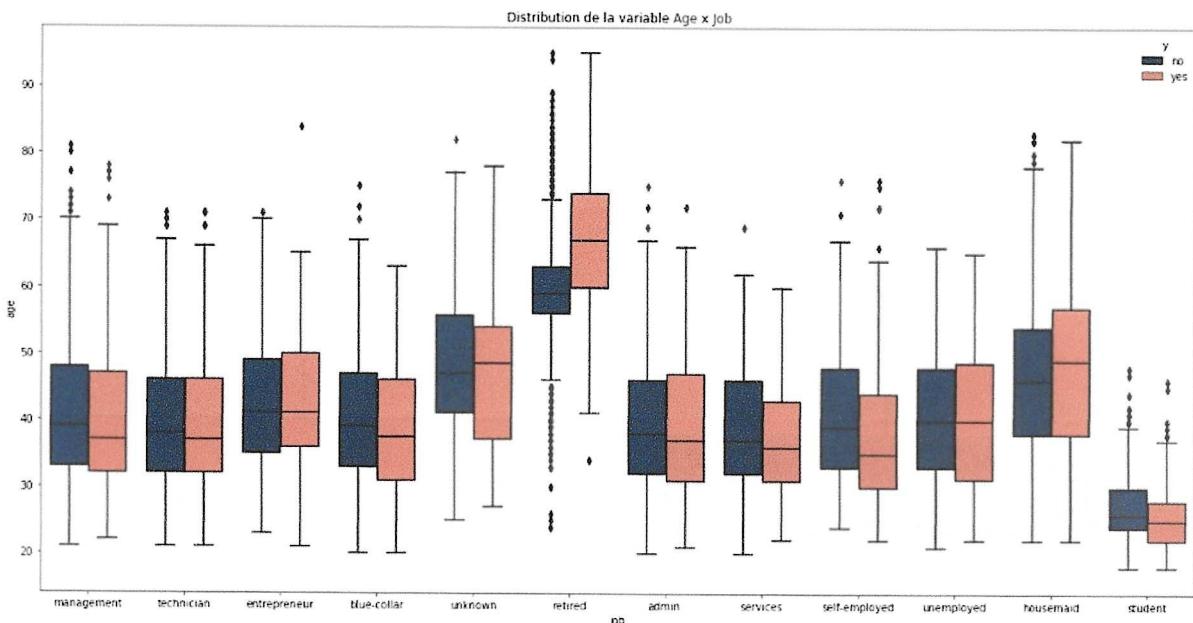
Ce rejoins l'observation que j'ai faite précédemment – en effet les métiers observés plus favorables à la souscription du produit correspondent aux mêmes secteurs d'activités.

Variable : « job » vs « age » : indique le métier du client et indique son âge. Variables catégorielles.

Objectif : Analyser la répartition des différents métiers des clients, ainsi que la distribution de l'âge par rapport à la variable cible.

```
# Création de la figure
fig = plt.figure(figsize = (20, 10))

# Création graphique
sns.set_palette(custom_palette)
sns.boxplot(x = "job", y = "age", hue = "y", data = df)
plt.title("Distribution de la variable Age x Job"); # Ajout d'un titre;
```



Je remarque que l'ensemble des métiers sont équilibrés sur la répartition de l'âge entre les personnes par rapport à la variable cible. Sauf pour les retraités, où je remarque que les jeunes retraités ont tendance à refuser le produit de placement financier.

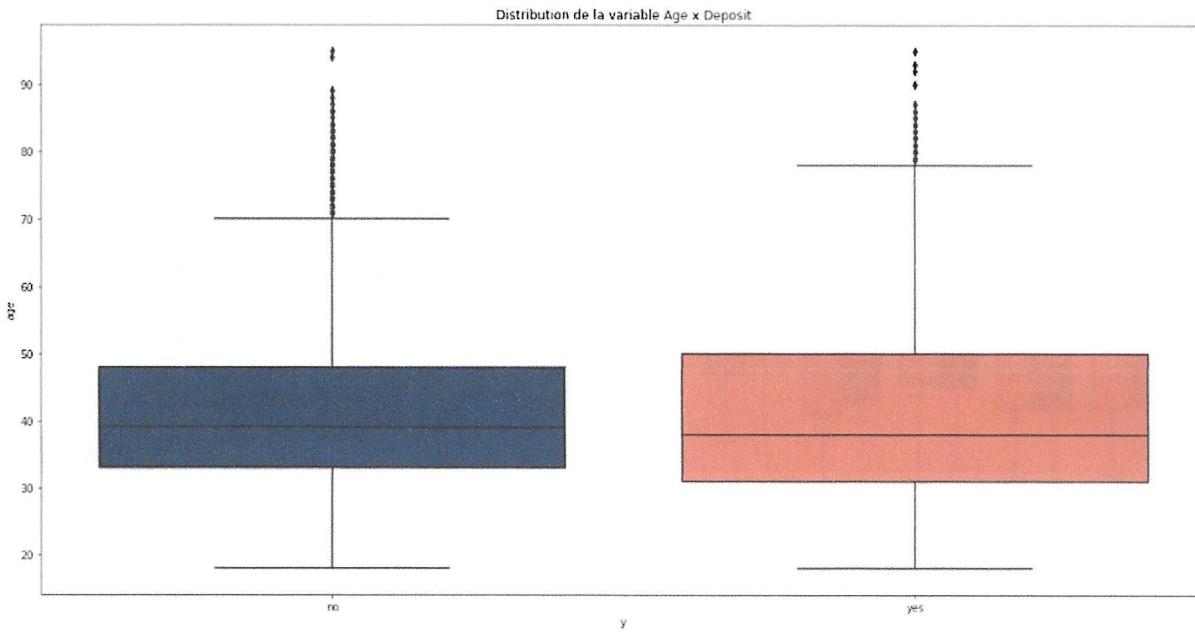
On peut également observer que l'âge est réparti entre 30 et 50 ans pour les personnes actives. Je pense qu'il s'agit de la tranche d'âge où on est susceptibles de vouloir faire des projets demandant un financement particulier.

Variable : cible vs « age » : indique l'acceptation du produit par le client et son âge. Variables catégorielles.

Objectif : Analyser la répartition de l'âge par rapport à la variable cible. Afin de savoir si le produit intéresse des personnes âgées ou jeunes.

```
# Création de La figure
fig = plt.figure(figsize = (20, 10))

# Création graphique
sns.set_palette(custom_palette)
sns.boxplot(x = "y", y = "age", data = df)
plt.title("Distribution de la variable Age x Deposit");# Ajout d'un titre;
```



La distribution est équilibrée principalement entre 35 et 49 ans pour les refus et entre 30 et 50 ans pour les acceptations.

Comme l'observation précédente, on est sur la tranche d'âge où on est susceptible de construire un projet d'avenir, épargner dans ce cas est une bonne solution pour préparer un futur investissement.

Gaetan GAUTHIER RICHEBOURG  
PROJET DATASCIENTEST : BANK MARKETING

L'enjeu est de savoir quelles informations concernant les personnes contactées, permettent de déterminer le choix du client à souscrire ou non au produit de placement. J'ai décidé de garder l'ensemble de toutes les variables disponibles afin d'alimenter au mieux le modèle de machine learning.

Mais pour pouvoir utiliser l'ensemble des données disponibles dans un modèle, il faut convertir les variables de type objet en variable de type numérique.

Je vais donc procéder de la manière suivante pour les variables de type objet :

- La plupart des variables de type objet dans le jeu de donnée, sont des informations binaires « yes » ou « no », par conséquent je décide de convertir la modalité « yes » par « 1 » et « no » par « 0 ».
- Il y a six variables avec plus de deux modalités, dans la continuité de ce que j'ai fait précédemment, je décide de remplacer les modalités par une suite numérique.

```
# Remplacement dans la variable 'deposit' : 'no' = '0' et 'yes' = '1'
df['y'] = df['y'].replace(to_replace = ['yes', 'no'], value = [1, 0])

# Remplacement dans la variable 'month' : par un chiffre
df['month'] = df['month'].replace(to_replace = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct',
                                                'nov', 'dec'],
                                  value = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

# Remplacement dans la variable 'job' : par un chiffre
df['job'] = df['job'].replace(to_replace = ['admin.', 'technician', 'services', 'management', 'retired', 'blue-collar',
                                              'unemployed', 'entrepreneur', 'housemaid', 'unknown', 'self-employed', 'student'],
                               value = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

# Remplacement dans la variable 'marital' : par un chiffre
df['marital'] = df['marital'].replace(to_replace = ['married', 'single', 'divorced'], value = [1, 2, 3])

# Remplacement dans la variable 'education' : par un chiffre
df['education'] = df['education'].replace(to_replace = ['secondary', 'tertiary', 'primary', 'unknown'], value = [1, 2, 3, 4])

# Remplacement dans la variable 'default' : par un chiffre
df['default'] = df['default'].replace(to_replace = ['yes', 'no'], value = [1, 0])

# Remplacement dans la variable 'housing' : par un chiffre
df['housing'] = df['housing'].replace(to_replace = ['yes', 'no'], value = [1, 0])

# Remplacement dans la variable 'loan' : par un chiffre
df['loan'] = df['loan'].replace(to_replace = ['yes', 'no'], value = [1, 0])

# Remplacement dans la variable 'contact' : par un chiffre
df['contact'] = df['contact'].replace(to_replace = ['unknown', 'cellular', 'telephone'], value = [1, 2, 3])

# Remplacement dans la variable 'poutcome' : par un chiffre
df['poutcome'] = df['poutcome'].replace(to_replace = ['unknown', 'other', 'failure', 'success'], value = [1, 2, 3, 4])

# Aperçu du DataFrame
df.head()
```

Aperçu des cinq premières lignes du DataFrame une fois les remplacements effectués :

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	4	1	2	0	2143	1	0	1	5	5	261	1	-1	0	1	0
1	44	2	2	1	0	29	1	0	1	5	5	151	1	-1	0	1	0
2	33	8	1	1	0	2	1	1	1	5	5	76	1	-1	0	1	0
3	47	6	1	4	0	1506	1	0	1	5	5	92	1	-1	0	1	0
4	33	10	2	4	0	1	0	0	1	5	5	198	1	-1	0	1	0

## IV. DES TESTS STATISTIQUES

### a. La corrélation de Pearson

Pour compléter la visualisation de la donnée, je décide de faire quelques tests statistiques pour commencer à chercher des corrélations entre la variable cible et les différentes variables disponibles. Pour ça j'utilise le test statistique de la corrélation de Pearson.

#### ❖ Corrélation entre la variable cible et la variable « campaign »

```
from scipy.stats import pearsonr
pd.DataFrame(pearsonr(df['y'], df['campaign']), index=['pearson_coeff', 'p-value'], columns=['resultat_test'])
```

	resultat_test
pearson_coeff	-7.317201e-02
p-value	1.012347e-54

Il y a peu de corrélation entre les variables.

#### ❖ Corrélation entre la variable cible et la variable « age »

```
pd.DataFrame(pearsonr(df['y'], df['age']), index=['pearson_coeff', 'p-value'], columns=['resultat_test'])
```

	resultat_test
pearson_coeff	2.515502e-02
p-value	8.825644e-08

L'âge a une corrélation avec la variable cible.

#### ❖ Corrélation entre la variable « campaign » et la variable « age »

```
pd.DataFrame(pearsonr(df['campaign'], df['age']), index=['pearson_coeff', 'p-value'], columns=['resultat_test'])
```

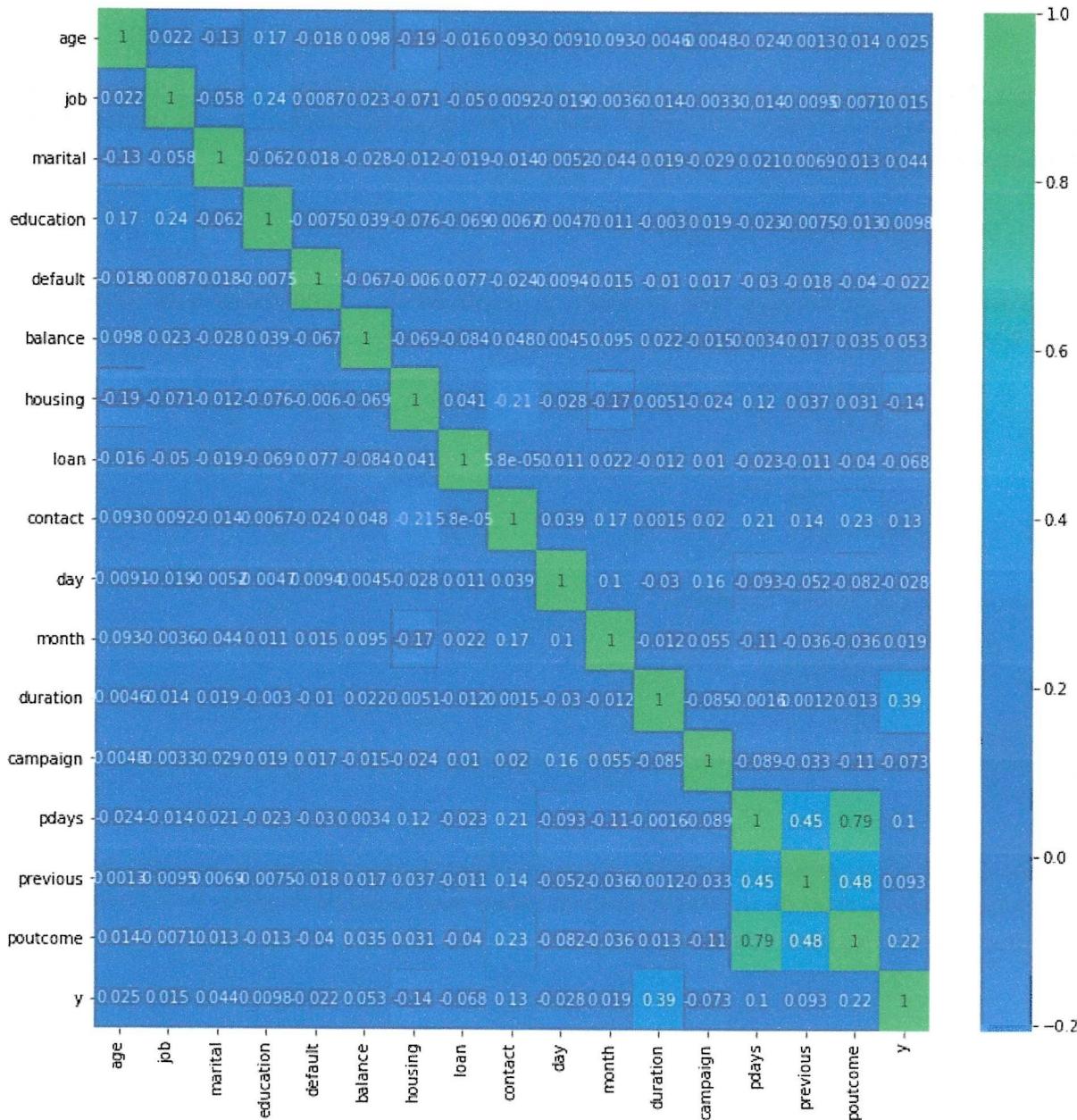
	resultat_test
pearson_coeff	0.004760
p-value	0.311463

Il y a peu de corrélation entre les variables.

### b. Heatmap

Pour illustrer toutes les corrélations entre les variables que nous avons, je décide d'utiliser une Heatmap :

```
cor=df.corr()
fig, ax = plt.subplots(figsize=(12,12))
sns.heatmap(cor, annot=True, ax=ax, cmap='winter');
```



La variable cible a une corrélation avec les variables : « poutcome », « duration », « pdays » et « contact ».

D'autres corrélations notamment : entre les variables « previous » et « poutcome », entre les variables « pdays » et « poutcome », entre les variables « pdays » et « previous ».

## V. MODÉLISATION

### a. Choix des variables

Afin de faire les tests statistiques, j'ai modifié l'ensemble des variables disponibles de type objet en type numérique. Grâce à cette modification je peux utiliser toutes les données à ma disposition dans des modèles de machine learning.

Par conséquent je décide de conserver l'ensemble des données à ma disposition.

### b. Préparation des données

Les algorithmes de Machine Learning de Python ne peuvent s'appliquer que sur des données de type numérique et ayant une valeur (les valeurs nulles ou manquantes doivent être soit supprimées du DataSet, soit remplacer par une valeur par défaut).

### c. Stockage de la variable cible

Afin d'éviter le surapprentissage dans un modèle supervisé (où nous connaissons la variable cible) je stock la variable cible du jeu de donnée.

```
#Stockage de la variable cible
data_ml = df.drop(columns ='y')
y = df['y']
```

### d. Echantillonnage

Je découpe le jeu de donnée comme suit :

- 80% pour le jeu d'entraînement
- 20% pour le jeu de test

```
#Séparation du jeu de données
X_train, X_test, y_train, y_test = train_test_split(data_ml, y, test_size = 0.2)
```

La suite c'est d'utiliser différents modèles de machine learning afin d'étudier les résultats de chacun, pour ensuite faire le choix du résultat le plus intéressant pour le jeu de données. Puis d'optimiser le modèle en ajustant les hypers paramètres.

## VI. MACHINE LEARNING

Le jeu de donnée étant déséquilibré sur la variable cible, il faut prendre en considération que le modèle sera influencé par le déséquilibre de la variable cible. Pour gérer les problèmes de Classification déséquilibrée, on peut utiliser différentes méthodes que je ne vais pas détailler ici.

### a. K les plus proche voisins & l'Oversampling

Dans mon cas je décide d'utiliser une méthode de sur-échantillonnage : Oversampling. Le but étant d'augmenter le nombre d'observations des classes minoritaires afin d'arriver à un ratio classe minoritaire/ classe majoritaire satisfaisant.

Oversampling pour équilibrer mon jeu de données :

```
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import RandomOverSampler

# Sur-échantillonnage
rOs = RandomOverSampler()
X_ro, y_ro = rOs.fit_resample(X_train, y_train)
# Entrainement du modèle de régression logistique
lr = LogisticRegression()
lr.fit(X_ro, y_ro)
# Affichage des résultats
y_pred = lr.predict(X_test)
```

Premier modèle de machine learning KNN :

```
for k in range(2, 50):
    knn = neighbors.KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_ro, y_ro)

#Prédictions
y_pred_knn = knn.predict(X_test)

#Résultats
display(pd.crosstab(y_test, y_pred_knn, rownames=['Classes réelles'], colnames=['Classes prédictes']))

#Score
display(knn.score(X_test, y_test))
```

Résultat :

		Classes prédictes	0	1
		Classes réelles		
		0	6101	1868
		1	267	807
0.763905783478934				

J'obtiens un bon score après avoir effectué l'Oversampling.

## b. SVM pénalisé

Le SVM pénalisé consiste à utiliser une fonction du modèle SVM, qui va directement utiliser un échantillon des données disponibles, pour équilibrer les classes.

```
svm = svm.SVC(gamma ='scale', class_weight= "balanced")  
  
svm.fit(X_train, y_train)  
# Résultats  
svm_preds = svm.predict(X_test)  
display(pd.crosstab(y_test, svm_preds, rownames=['Classe réelle'], colnames=['Classe prédictive']))
```

Résultat :

		Classe prédictive	0	1
		Classe réelle		
		0	6336	1667
		1	313	727

Score obtenu :

	precision	recall	f1-score	support
0	0.95	0.79	0.86	8003
1	0.30	0.70	0.42	1040
accuracy			0.78	9043
macro avg	0.63	0.75	0.64	9043
weighted avg	0.88	0.78	0.81	9043

Les scores obtenus sont intéressants avec 0.78 d'accuracy.

Pourtant, après avoir échangé avec mon mentor et voulant compléter le tout avec de l'interprétabilité, je découvre un autre modèle de machine learning qui m'a inspiré pour la suite du projet.

### c. XGBoost

Le modèle de Machine Learning XGBoost est considéré comme un « black-box » par beaucoup étant difficilement interprétable, je vais tout de même essayer dans mon projet d'utiliser cette méthode, de me challenger et d'interpréter les résultats.

Pour commencer je définie un pipeline pour pouvoir appliquer un GridSearch pour définir les hyperparamètres les plus adéquats pour utiliser le modèle.

Dans mon pipeline j'intègre également un score d'erreur quadratique moyenne, afin de pouvoir augmenter l'efficacité du choix des hyperparamètres. Ça consolide les projections de prédictions qui sont faites afin d'optimiser au mieux le modèle.

```
def algorithm_pipeline(X_train, X_test, y_train, y_test,
                      model, param_grid, cv=10, scoring_fit='neg_mean_squared_error',
                      do_probabilities = False):
    gs = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=cv,
        n_jobs=-1,
        scoring=scoring_fit,
        verbose=2
    )
    fitted_model = gs.fit(X_train, y_train)

    if do_probabilities:
        pred = fitted_model.predict_proba(X_test)
    else:
        pred = fitted_model.predict(X_test)

    return fitted_model, pred
```

Ensuite je cherche les hyperparamètres les plus adaptés à mon modèle.

```
model = xgb.XGBClassifier()
param_grid = {'subsample':[0.5, 0.75, 1],
              'objective':['binary:logistic'],
              'colsample_bytree':[0.7, 1],
              'max_depth':[2, 6, 12],
              'min_child_weight':[1,5,15],
              'learning_rate':[0.3, 0.1, 0.03],
              'n_estimators':[100]}

model, pred = algorithm_pipeline(X_train, X_test, y_train, y_test, model, param_grid, cv=5)
print(model.best_params_)

Fitting 5 folds for each of 162 candidates, totalling 810 fits
{'colsample_bytree': 1, 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5, 'n_estimators': 100, 'objective': 'binary:logistic', 'subsample': 0.75}
```

Les hyperparamètres retenus :

colsample\_bytree : 1  
learning\_rate : 0.1  
max\_depth : 6  
min\_child\_weight : 5  
subsample : 0.75

J'entraîne mon modèle avec les hyperparamètres retenus.

```
# Paramètres d'entraînement
params = {'colsample_bytree': 1, 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 5,
          'objective': 'binary:logistic', 'subsample': 0.75}

# Conversion des jeux de données en DMatrix
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(data = X_test, label = y_test)

# Entraînement du modèle
data_xg = xgb.train(params, dtrain)
```

Score obtenu

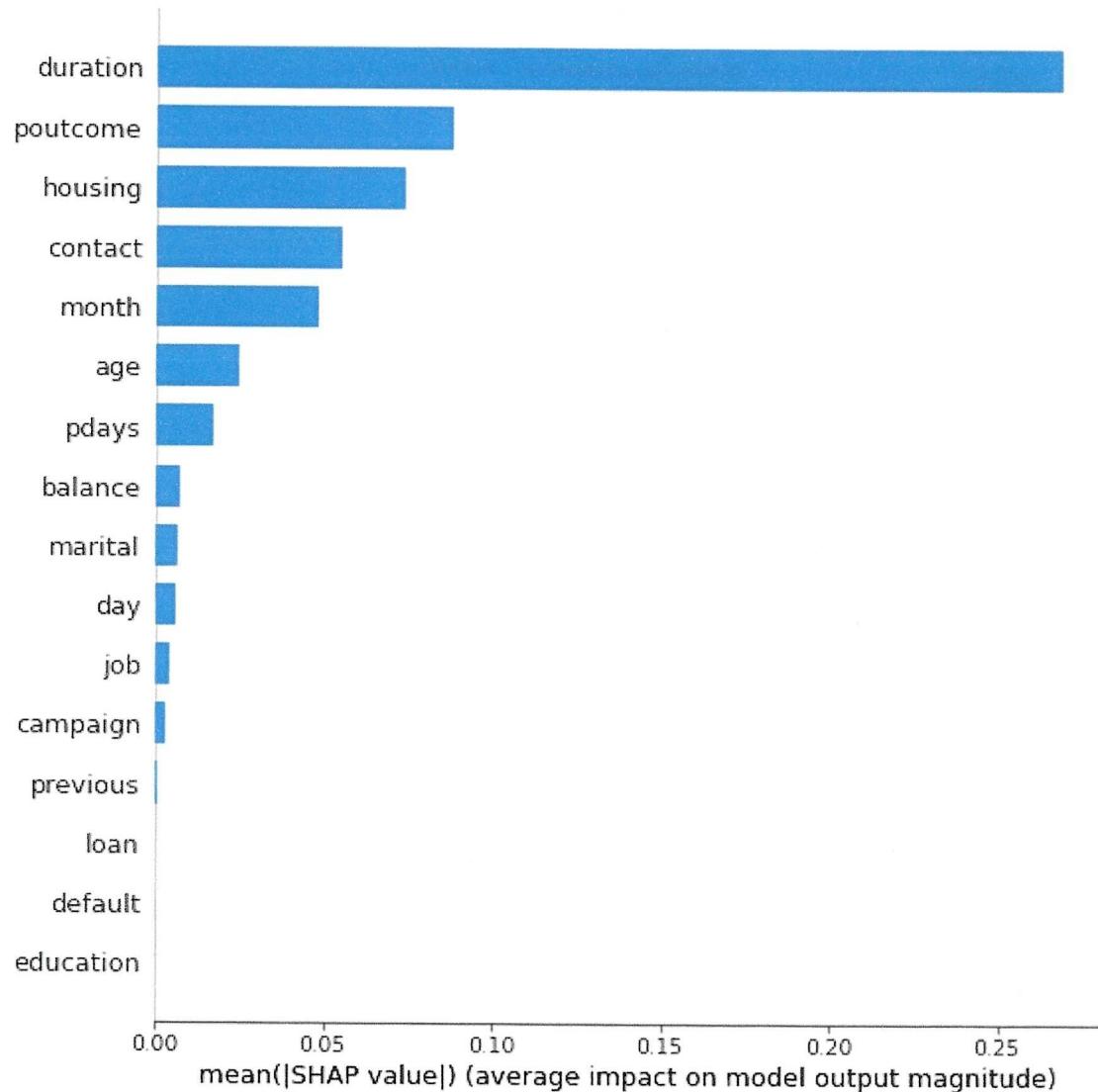
	precision	recall	f1-score	support
0	0.92	0.98	0.95	7983
1	0.68	0.36	0.47	1060
accuracy			0.90	9043
macro avg	0.80	0.67	0.71	9043
weighted avg	0.89	0.90	0.89	9043

J'obtiens un score plus élevé que précédemment et intéressant. C'est pour cette raison que j'ai choisis d'aller jusqu'à l'interprétabilité avec le modèle XGBoost.

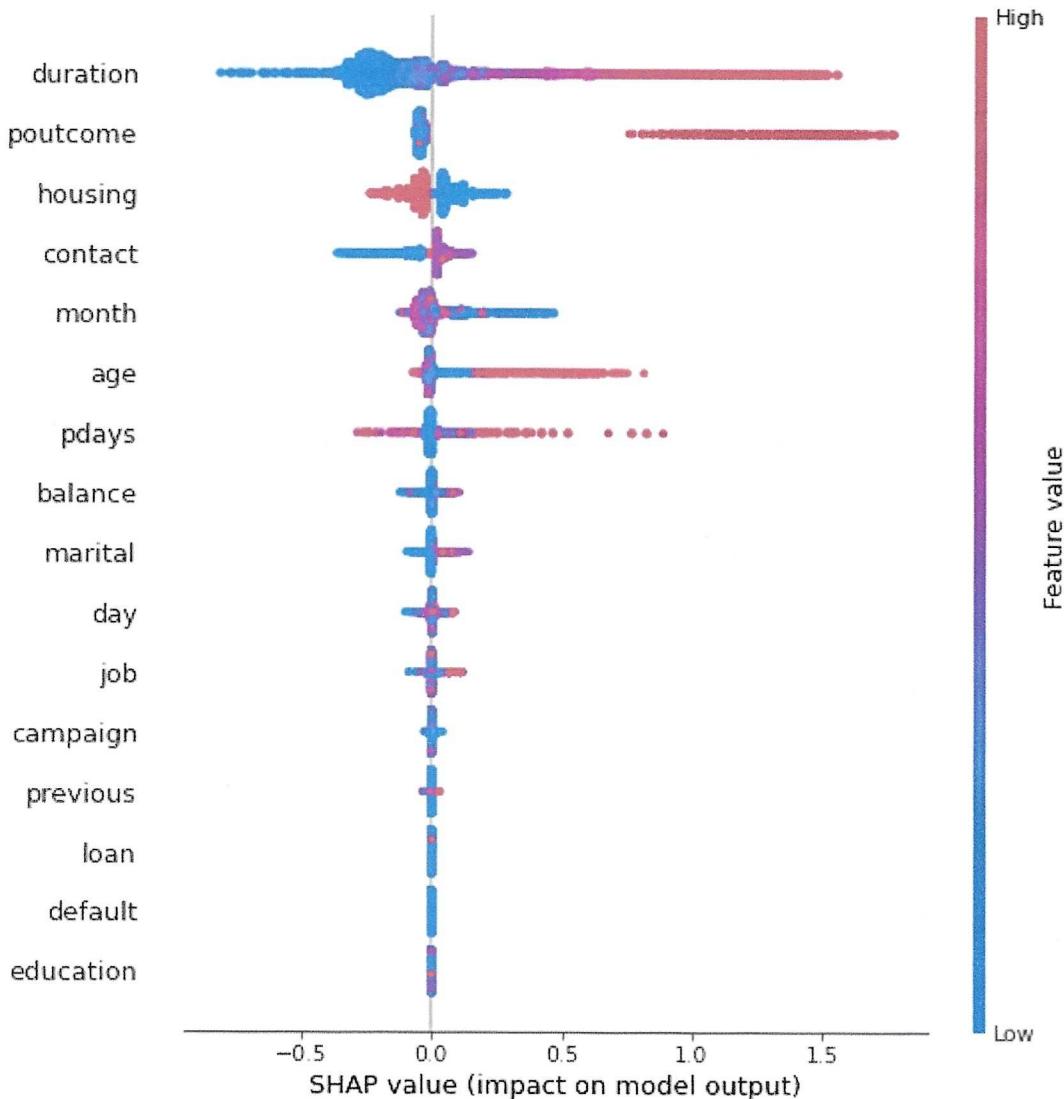
## VII. L'INTERPRÉTABILITÉ

L'idée est donc d'expliquer, comment une variable peut avoir un impact sur la décision que le modèle va prendre pour faire son choix de Classification. J'ai décidé d'utiliser SHAP qui va permettre d'avoir une approche globale, puis une approche individuelle.

### a. Approche globale



Je constate que la variable « duration » a un impact élevé sur le modèle, plus important que les autres variables. Puis on observe les variables « poutcome », « housing », « contact » et « month » avec un impact sur le modèle plus modéré que la variable « duration ».



On peut voir que la variable « duration » a une forte influence sur le modèle, suivie des variables « poutcome », « housing » et « contact ».

Ça signifie que si la durée du dépôt est importante, alors la personne aura tendance à souscrire. On peut facilement conclure que si le prospect est contacté il y a plus de chance qu'il souscrit au produit.

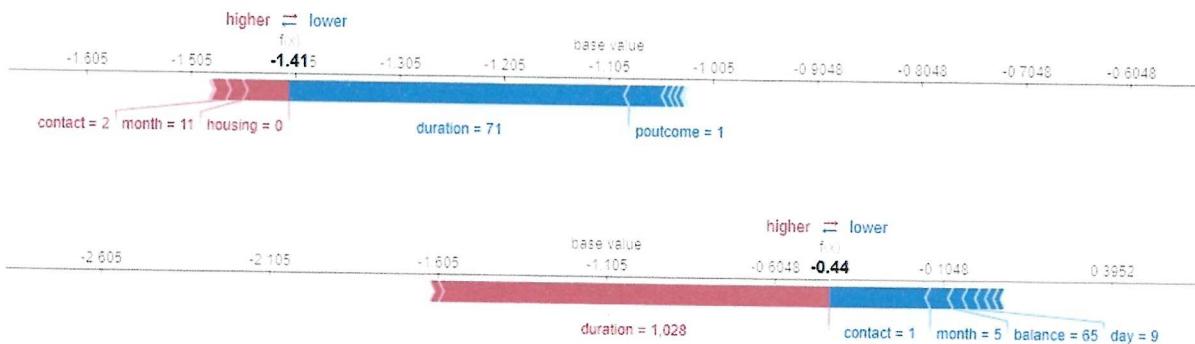
Dans le second graphique ce qui est intéressant, c'est que si le prospect n'est pas propriétaire il est plus susceptible de souscrire au produit. J'observe cela grâce à la variable « housing ».

Avec SHAP nous pouvons également faire une approche individuelle – en effet cela permet d'identifier un résultat de notre modèle et d'analyser l'impact que chaque variable a eu sur ce résultat seulement.

## b. Approche individuelle

L'approche individuelle permet de comparer deux individus pour voir comment les variables influent sur le modèle.

```
shap.initjs()  
  
display(shap.force_plot(explainer.expected_value, shap_values[12,:], X_test.iloc[12,:]))  
display(shap.force_plot(explainer.expected_value, shap_values[84,:], X_test.iloc[84,:]))
```



Le premier exemple, l'individu n° 12 on retrouve nos variables « housing » et « contact » qui influent à la hausse sur le choix du modèle, puis les variables « duration » et « poutcome » qui influent à la baisse sur le choix du modèle.

Le second exemple, l'individu n° 84 on retrouve nos variables « duration » qui influent fortement à la hausse, puis les variables « contact » et « month » qui influent à la baisse le choix du modèle.

## VIII. Conclusion

Je peux recommander à l'établissement de privilégier des placements financiers long, pour des personnes qui ne sont pas propriétaires, d'un âge entre 30 et 50 ans. Les produits de type : PEL ou CEL sont ceux, à ma connaissance, qui seront les plus intéressant à proposer aux prospects au vu de l'interprétabilité du modèle utilisé.

Il semblerait également, qu'il faille contacter les personnes plus d'une fois pour que celles-ci souscrivent au produit de placement financier. On peut imaginer, que les prospects ont besoin de temps de réflexion ou d'accompagnement avant de souscrire au produit.

Grâce à ce projet j'ai pu mettre en pratique une bonne partie de ce que j'ai appris lors de ma formation et pouvoir utiliser mes compétences par la suite dans le monde professionnel.