

▼ Rendu du projet : ACTUARIAT ET DATA-SCIENCE

Ibrahim EZ-ZAHRAOUI

Third-pa

▼ 0. Importation des librairies

```
# Installer le package catboost
!pip install catboost

# Installer ipywidgets Python package
!pip install ipywidgets

# Activer l'extension de l'interface
!jupyter nbextension enable --py widgetsnbextension

# Scikit-optimize pour l'optimisation
!pip install scikit-optimize

# shap pour l'importance des features
!pip install shap

Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.6.1->notebook)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.1->jupyter-core)
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->notebook>=4.4.1->notebook)
Requirement already satisfied: lmfit in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: tinycc2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook>=4.4.1->notebook)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-packages (from nbformat->notebook>=4.4.1->notebook)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat->notebook>=4.4.1->notebook)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat)

Echec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. Voir diff. (from jsonschema>=2.6->nbformat)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert->notebook)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->notebook)
Enabling notebook extension jupyter-js-widgets/extension...
Paths used for configuration of notebook:
    /root/.jupyter/nbconfig/notebook.json
Paths used for configuration of notebook:

```

- Validating: OK

```
Paths used for configuration of notebook:
    /root/.jupyter/nbconfig/notebook.json
Requirement already satisfied: scikit-optimize in /usr/local/lib/python3.10/dist-packages (0.9.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.3.2)
Requirement already satisfied: pyyaml>=16.9 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (23.7.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.23.5)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.10.1)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from pyyaml>=16.9->scikit-optimize) (6.0.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->scikit-optimize)
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages (0.42.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.10.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.0)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (23.1)
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.10/dist-packages (from shap) (0.0.7)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.56.4)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.39.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from numba->shap) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2023.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (2.0.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
```

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.model_selection import train_test_split
from catboost import CatBoostRegressor, cv, Pool
import skopt
from numpy import mean
from sklearn.model_selection import cross_val_score
from skopt.space import Integer
from skopt.space import Real
from skopt.space import Categorical
from skopt.utils import use_named_args
from skopt import gp_minimize
from sklearn.metrics import mean_squared_error
import shap
shap.initjs()

```



▼ 1. Données

▼ 1.1. Importation des données

```

# Importation des données
train = pd.read_csv("/content/drive/MyDrive/Study case/labeled_dataset.csv", sep = ';')
submission = pd.read_csv("/content/drive/MyDrive/Study case/scoring_dataset.csv", sep = ';')

train.head()

```

index	Age	Prime mensuelle	Categorie socio professionnelle	Kilometres parcourus par mois	Coefficient bonus malus	Type de vehicule	Score CRM	Niveau de vie	Marque	Salaire annuel	Score credit	entre' an
0	0	58.0	40.0	Etudiant	973	106	SUV	164	3762	Peugeot	20420	309
Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet.												
1	3	22.0	8.0	Etudiant	771	90	5 portes	111	2500	Renault	6790	786
2	5	25.0	18.0	Sans emploi	372	123	5 portes	160	2058	Peugeot	15140	320
3	5	25.0	18.0	Sans emploi	372	123	5 portes	160	2058	Peugeot	14400	850
4	6	54.0	22.0	Cadre	981	86	5 portes	148	3906	Renault	27350	895

```

print(f"Nombre de lignes d'entraînement : {len(train)}")
print(f"Nombre de lignes de prédiction : {len(submission)}")

```

```

Nombre de lignes d'entraînement : 1000
Nombre de lignes de prédiction : 300

```

▼ 1.2. Description des données

Ce jeu de données correspond à des données relatifs l'assurance.

Il est intéressant de noter qu'il a été intégralement généré. Donc, **les relations entre les variables ou leurs valeurs pourraient ne pas être significatives ou reflétante de la réalité.**

On définit quelques variables :

- **index:** Un identifiant unique pour chaque entrée dans le dataset.
- **Prime mensuelle:** la prime (potentiellement commerciale) d'assurance que l'assuré paie
- **Coefficient bonus malus :** reflète le comportement de conduite de l'assuré. Un bonus malus grand reflète une conduite imprudente.
- **Score CRM :** pareil que le bonus malus. Plus il est élevé plus la conduite de l'assuré est imprudente.
- **Score crédit :** évalue la solvabilité financière de l'assuré. Un score de crédit élevé indique une probabilité plus faible de défaut de paiement.
- **Cout entretien annuel :** du véhicule
- **Benefice net annuel :** le profit annuel généré par l'assuré pour l'assureur

Le but du projet est de prédire Le bénéfice net annuel étant donné un profil client spécifique. Il s'agit d'un problème de **régression**.

```
# Cible de prédiction
target = 'Benefice net annuel'
```

▼ 1.3. Qualité des données

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   index            1000 non-null    int64  
 1   Age              982 non-null    float64 
 2   Prime mensuelle  988 non-null    float64 
 3   Categorie socio professionnelle 1000 non-null    object  
 4   Kilometres parcourus par mois     1000 non-null    int64  
 5   Coefficient bonus malus        1000 non-null    int64  
 6   Type de vehicule          1000 non-null    object  
 7   Score CRM             1000 non-null    int64  
 8   Niveau de vie          1000 non-null    int64  
 9   Marque              947 non-null    object  
 10  Salaire annuel       1000 non-null    int64  
 11  Score credit         1000 non-null    int64  
 12  Cout entretien annuel 1000 non-null    int64  
 13  Benefice net annuel   1000 non-null    object  
dtypes: float64(2), int64(8), object(4)
memory usage: 109.5+ KB
```

▼ 1.3.1. Erreurs

On remarque que la variable **Benefice net annuel** est de dtype object à la place de float64. Ceci pourrait être du à une erreur lors de la génération du dataset. En effet, certaines valeurs contiennent ';' comme séparateur décimal. On procède à la correction.

```
# Remplacer les ',' par des '.'
train["Benefice net annuel"] = train["Benefice net annuel"].apply(lambda x : float(x.replace(',', '.')))
```

▼ 1.3.2. Valeurs manquantes

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

Le dataset contient des valeurs manquantes pour trois variables. Il est nécessaire de les traiter afin d'éviter des biais dans l'analyse ou bien des pertes d'information.

```
# Représenter les valeurs manquantes dans le dataset
plt.figure(figsize=(12,4))
sns.heatmap(train.isnull(), cbar=False, cmap='viridis', yticklabels=False)
plt.title('Missing value in the dataset')
```

```
Text(0.5, 1.0, 'Missing value in the dataset')
```

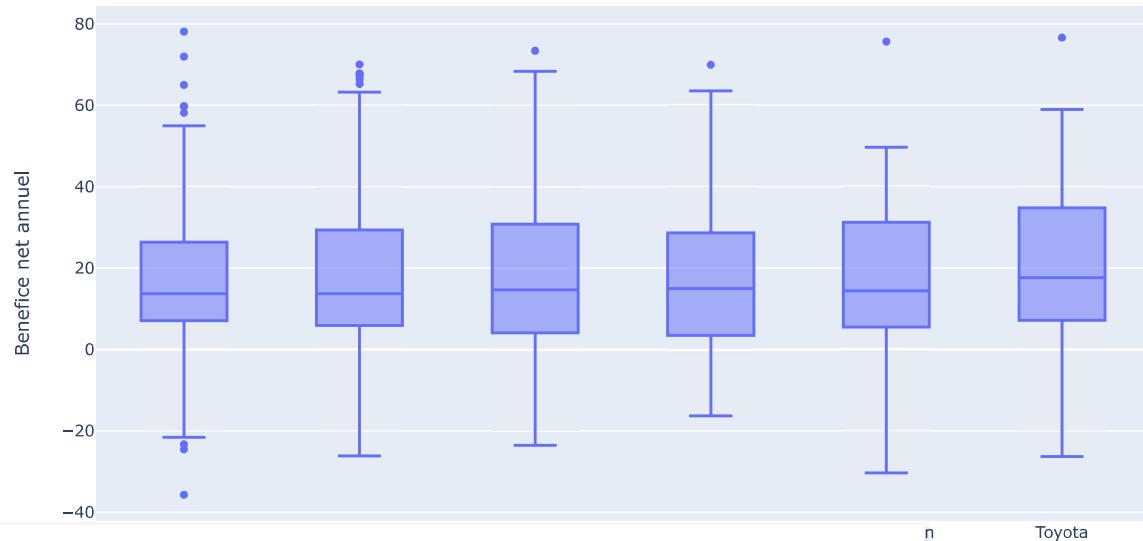
Missing value in the dataset

Pour prendre la décision de comment on peut remplir les champs vide ou de la possibilité de les supprimer, on regarde la relation de la variable avec la cible.

Variable Marque :

On regarde la distribution de notre target par rapport aux différentes instances de Marque.

```
# Distribution de la cible par rapport à la marque
fig = px.box(train, x="Marque", y="Benefice net annuel")
fig.show()
```



Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

Nous remarquons que la marque de la voiture n'a pas d'impact significatif sur le bénéfice net annuel. On peut pour le moment se contenter de renseigner les champs vides par une valeur aléatoire. Le choix de cette valeur dépend de la probabilité d'occurrence de la catégorie dans le dataset.

```
# le nombre de champs vides dans la colonne Marque
num_missing = train['Marque'].isnull().sum()

# Décompte d'occurrence de chaque instance dans le dataset
categories = train['Marque'].value_counts().to_dict()
print(categories)

# Calcul de probabilité d'occurrence
length= len(train)
cat_prob = {key : value/(length - num_missing) for key, value in categories.items()}

# Génération des échantillons aléatoires pour remplir les champs vides
random_samples = np.random.choice(list(categories.keys()), size=num_missing, p=list(cat_prob.values()))

# Remplacement des champs vides par les échantillons aléatoires générés
train.loc[train['Marque'].isnull(), 'Marque'] = random_samples

{'Peugeot': 269, 'Renault': 238, 'Citroen': 195, 'Volkswagen': 106, 'Toyota': 89, 'Opel': 50}
```

Variables Age et Prime mensuelle :

Il y a 18/1000 lignes avec la variable age inconnue.

```
len(train)
```

1000

```

missing_age = train[train["Age"].isnull()]
missing_prime_mensuelle = train[train["Prime mensuelle"].isnull()]
without_missing_values = train[~(train["Prime mensuelle"].isnull()) & ~(train["Age"].isnull())]

fig = make_subplots(rows=2,
                     cols=2,
                     subplot_titles=("Dataset complet", "prime mensu & age /= null", "age = null", "prime mensuelle = null"),
                     x_title='Bénéfice',
                     y_title='Distribution (%)')

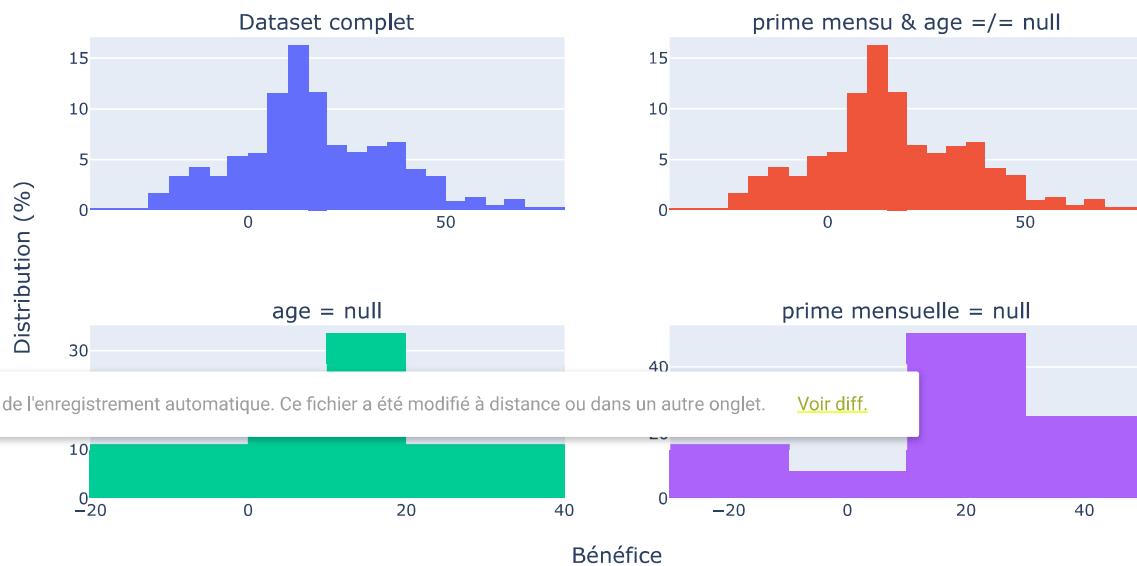
target_dist = go.Histogram(x=train["Benefice net annuel"], histnorm='percent')
target_dist_without_missing = go.Histogram(x=without_missing_values["Benefice net annuel"], histnorm='percent')
target_dist_missing_age = go.Histogram(x=missing_age["Benefice net annuel"], histnorm='percent')
target_dist_missing_prime_mensuelle = go.Histogram(x=missing_prime_mensuelle["Benefice net annuel"], histnorm='percent')

fig.add_trace(target_dist, row=1, col=1)
fig.add_trace(target_dist_without_missing, row=1, col=2)
fig.add_trace(target_dist_missing_age, row=2, col=1)
fig.add_trace(target_dist_missing_prime_mensuelle, row=2, col=2)

fig.update_layout(showlegend=False, title_text="Distribution du bénéfice")

```

Distribution du bénéfice



Nous remarquons que la suppression des lignes contenant des valeurs manquantes pour les variables **Prime mensuelle** et **Age** n'impacte pas la distribution de probabilité de la cible **Bénéfice annuel net**. Donc, on peut se permettre de supprimer les lignes avec des valeurs manquantes.

On a privilégié la suppression aux méthodes d'imputation par des statistiques agrégés (max, mmin, moyenne) afin de :

- Eviter d'augmenter la variabilité des données. Cela peut potentiellement rendre le modèle plus sensible aux valeurs aberrantes et affecter leur sensibilité.
- Eviter l'incohérence avec le processus génératif des données.
- 30 lignes soit 3% des données supprimées.

```
# Supprimer les lignes avec Prime mensuelle ou age manquants
train = train.dropna()
```

```
train.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 970 entries, 0 to 998
Data columns (total 14 columns):
 #   Column           Non-Null Count Dtype  
---  --  
0   index            970 non-null   int64  
1   Age              970 non-null   float64 
2   Prime mensuelle 970 non-null   float64 
3   Catégorie socio professionnelle 970 non-null object  
4   Kilometres parcourus par mois    970 non-null   int64  

```

```

5 Coefficient bonus malus      970 non-null    int64
6 Type de vehicule          970 non-null    object
7 Score CRM                  970 non-null    int64
8 Niveau de vie                970 non-null    int64
9 Marque                      970 non-null    object
10 Salaire annuel             970 non-null    int64
11 Score credit                 970 non-null    int64
12 Cout entretien annuel       970 non-null    int64
13 Benefice net annuel         970 non-null    float64
dtypes: float64(3), int64(8), object(3)
memory usage: 113.7+ KB

```

▼ 2. Statistiques descriptives

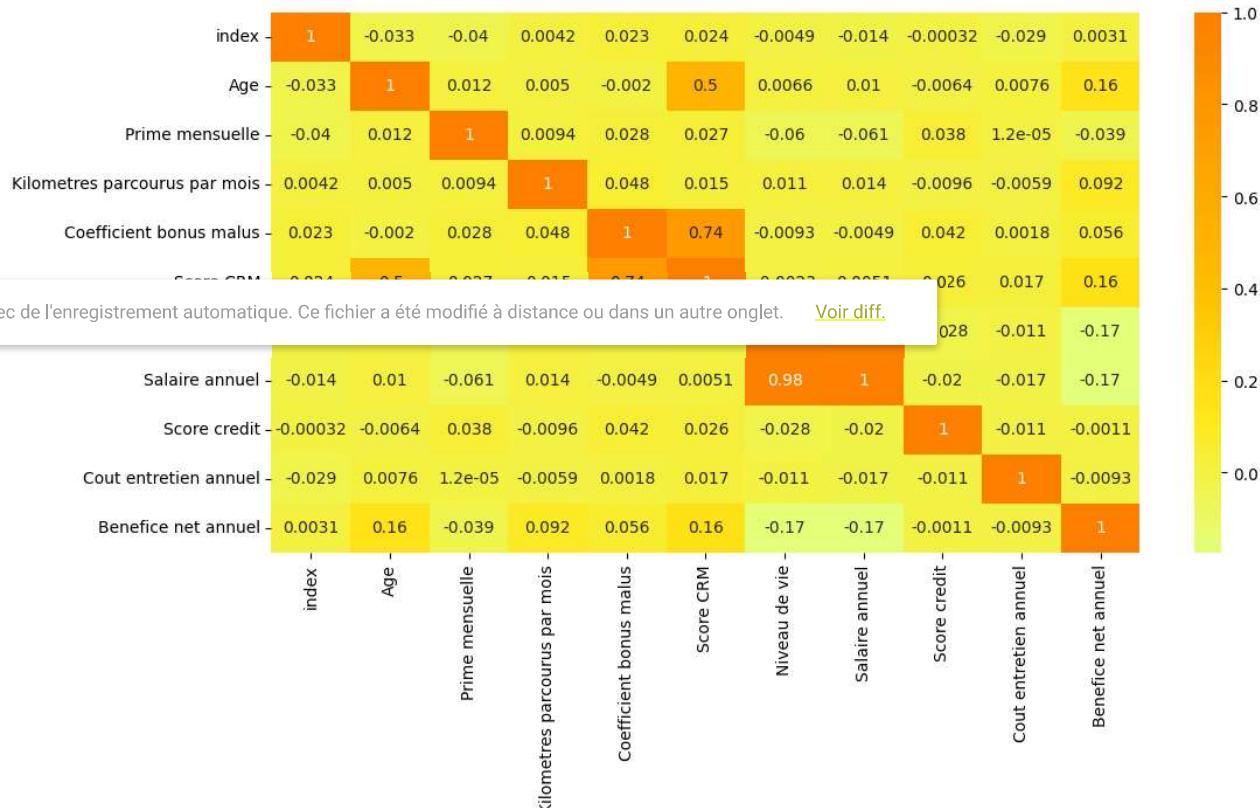
Il est intéressant de regarder les statistiques descriptives afin de résumer les données et identifier les tendances ainsi que les relations entre les variables.

▼ 2.1. Correlation

```
# Correlation Pearson entre les variables
plt.figure(figsize=(12,6))
corr = train.corr()
sns.heatmap(corr, cmap='Wistia', annot=True);
```

<ipython-input-22-db3c45333cf2>:3: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select on



Nous identifions **Salaire annuel** et **Niveau de vie** comme des variables fortement corrélées tout en ayant la même corrélation avec la variable cible. Nous pouvons supprimer la variable **Niveau de vie**.

```
# Supprimer la variable Niveau de vie
train.drop(columns = ['Niveau de vie', 'index'], inplace = True)
```

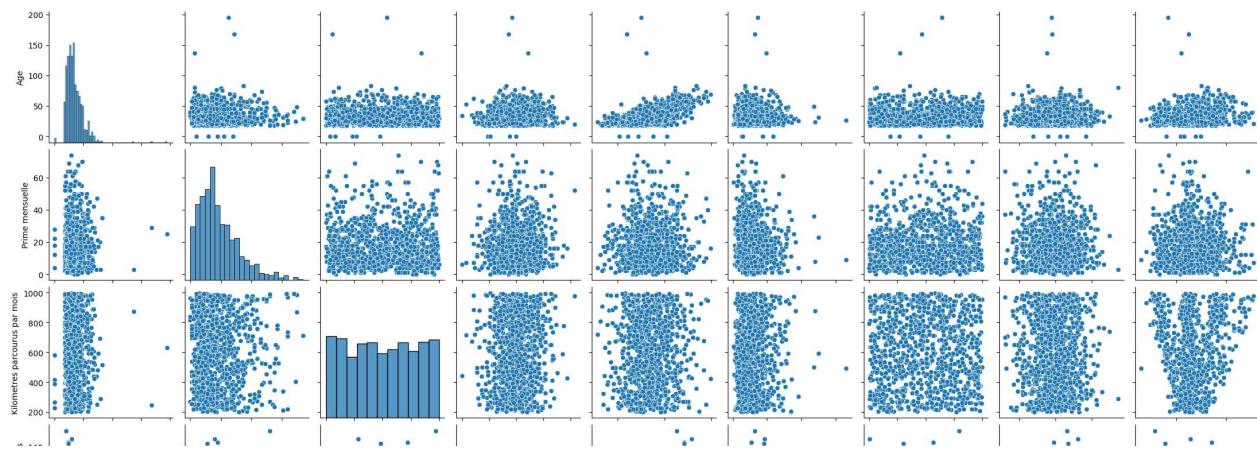
Il y a d'autres variables corrélées, mais vu qu'on n'est pas sur un grand volume de données, n peut se contenter de les laisser.

▼ 2.4. Distributions

Maintenant, nous visualisons la distribution des variables **numériques** ainsi que leurs relations avec la cible.

```
sns.pairplot(train[[col for col in train.columns]])
plt.show()
```

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

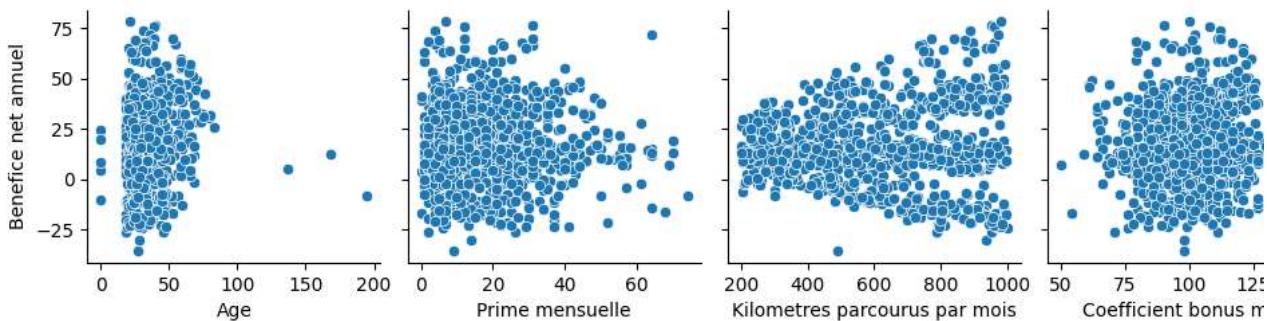


Le plot ci-dessus permet de visualiser les différentes distributions entre les variables **numériques**. Notamment les distributions de la variable cible par rapport aux autres.



Distribution de la variable cible par rapport aux variables numériques

```
# Distribution de la variable cible par rapport aux variables numériques
sns.pairplot(train[[col for col in train.columns if col != 'index']], y_vars = ["Benefice net annuel"])
plt.show()
```



Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet.

[Voir diff.](#)

Distribution de la variable cible par rapport aux variables catégorielles

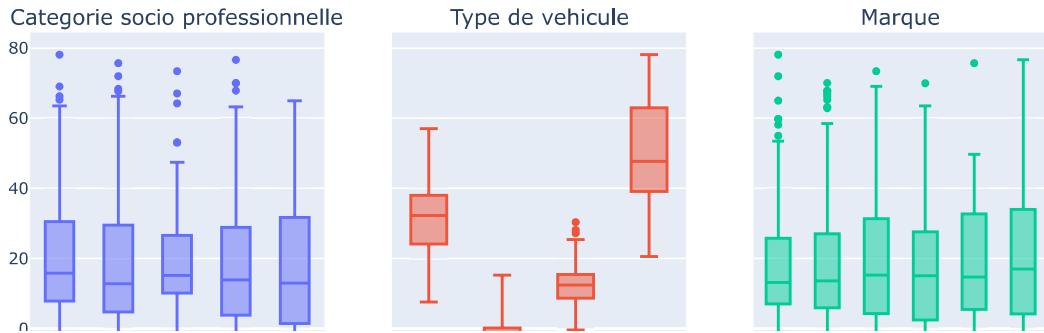
```
# Définir les colonnes catégorielles
cat_columns = train.select_dtypes(include=['object']).columns.tolist()

# Initier une figure
fig = make_subplots(rows=1,
                     cols=len(cat_columns),
                     subplot_titles=(cat_columns[0], cat_columns[1], cat_columns[2]),
                     shared_yaxes=True)

# Boxplots
col1 = go.Box(x = train[cat_columns[0]], y=train[target], name =cat_columns[0])
col2 = go.Box(x = train[cat_columns[1]], y=train[target], name =cat_columns[1])
col3 = go.Box(x = train[cat_columns[2]], y=train[target], name =cat_columns[2])

# Ajout des traces dans la figures
fig.add_trace(col1, row = 1, col = 1)
fig.add_trace(col2, row = 1, col = 2)
fig.add_trace(col3, row = 1, col = 3)

# Visualisation
fig.update_layout(showlegend = False)
fig.show()
```



Nous remarquons que **Type de véhicule** est une variable discriminante pour le véhicule. Notamment, le bénéfice est plus important avec les gens qui ont des véhicules utilitaire qu'avec les gens avec des véhicules à 3 portes.

▼ 3. Modèle

▼ 3.1. Encodage

Nous pouvons ensuite encoder les données catégorielles en utilisant des techniques tels que **One hot encoding**. Or, vu qu'on utilisera un modèle catboost, lui-même se chargera de l'encodage. Il est déconseillé de traiter les variables catégorielles à la main si on compte utiliser un modèle catboost.

▼ 3.2. Split des données

Nous splitons les données en données d'entraînement et données de test.

▼ 3.2.1 Split ordinaire

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

```
y = train[["Benefice net annuel"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Il est intéressant de faire attention à préserver la distribution de la variable cible quand on split les données afin de préserver la représentativité.

Pour examiner si la méthode de split ordinaire préserve la représentativité, on regarde la distribution de **y_test** en comparaison avec celle de **y_train**.

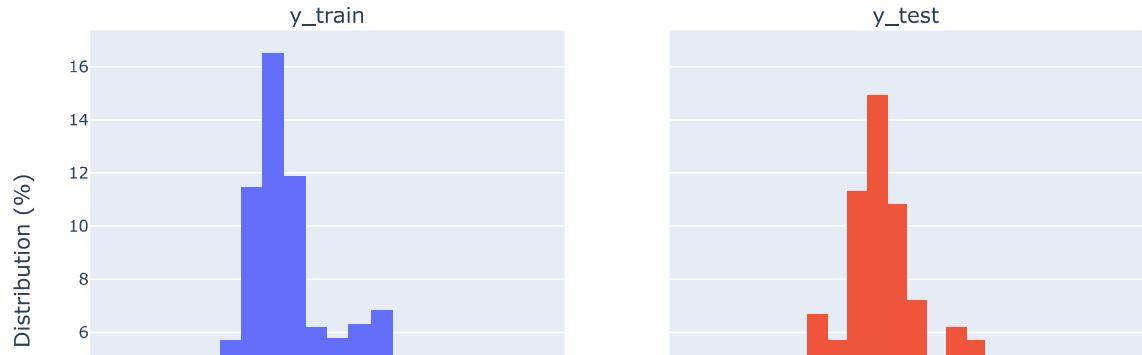
```
fig = make_subplots(rows=1,
                     cols=2,
                     subplot_titles=("y_train", "y_test"),
                     x_title='Bénéfice',
                     y_title='Distribution (%)',
                     shared_yaxes = True)

y_train_dist = go.Histogram(x=y_train["Benefice net annuel"], histnorm='percent')
y_test_dist = go.Histogram(x=y_test["Benefice net annuel"], histnorm='percent')

fig.add_trace(y_train_dist, row = 1, col = 1)
fig.add_trace(y_test_dist, row = 1, col = 2)

fig.update_layout(showlegend=False, title_text="distribution de y_train vs y_test")
```

distribution de y_train vs y_test



Nous remarquons qu'un split "normal" des données en données d'entraînement et données de test ne conserve pas la distribution de la variable cible.

Donc, nous allons remédier à ceci en utilisant une forme de stratification en créant des intervalles ou des catégories à partir des données numériques, puis en effectuant la séparation.

▾

3.2.2 Split stratifié

```
# Convertir y en intervalles/catégories
y_serie = y.iloc[:,0]

# min, max
y_min, y_max = min(y_serie), max(y_serie)

# spécifier les intervalles à stratifier. Ainsi, pour chaque intervalle de valeurs,
# nous prenons toujours une même proportion (ici 20%)
bins = [int(y_min) + 5*(i-1) for i in range(0, int(((y_max-y_min)/5+3))) if i!=1]
y_binned = pd.cut(y.iloc[:,0], bins=bins, labels=False)

#Effectuer la séparation entraînement-test tout en maintenant la distribution de y_binned
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y_binned)
```

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

```
# Visualiser la distribution
fig = make_subplots(rows=1,
                     cols=2,
                     subplot_titles=("y_train", "y_test"),
                     x_title='Bénéfice',
                     y_title='Distribution (%)',
                     shared_yaxes = True)

y_train_dist = go.Histogram(x = y_train["Benefice net annuel"], histnorm='percent')
y_test_dist = go.Histogram(x = y_test["Benefice net annuel"], histnorm='percent', xbins=dict(start=min(bins),end=max(bins),size=1))

fig.add_trace(y_train_dist, row = 1, col = 1)
fig.add_trace(y_test_dist, row = 1, col = 2)

fig.update_layout(showlegend=False, title_text="distribution de y_train vs y_test")
```

distribution de y_train vs y_test



Nous remarquons qu'avec la méthode de split stratifié, nous avons réussi à préserver la distribution.

3.3. Tuning des hyperparamètres

Catboost se charge du traitement des variables catégorielles. Il suffit de les lui identifier.

```
# Variables catégorielles
cat_features = train.select_dtypes(include=['object']).columns.tolist()
```

3.3.1. Pas d'apprentissage et Profondeur (optimisation bayésienne)

D'abord, nous optimisons le pas d'apprentissage (**learning_rate**) et la profondeur des arbres (**depth**) en parallèle afin de trouver une combinaison de ces deux hyperparamètres qui fonctionne bien ensemble et qui offre un bon équilibre entre la vitesse d'apprentissage et la capacité de modélisation.

Pour ce faire nous utilisons l'optimisation bayésienne grâce au module scikit-optimize. C'est une méthode qui consiste à utiliser les processus gaussiens qui généralisent le concept de loi normale aux fonctions. Après chaque expérience, l'algorithme identifie les points à fort potentiel qui correspondent au minimum et qui permettent d'explorer davantage d'intervalles.

Le modèle utilisé est **CatboostRegressor**. Pour chaque tuple de valeurs à tester donné, on procède à une **validation croisée** avec 4 folds. L'optimisation bayésienne à trouver **learning_rate** et **depth** qui minimisent la fonction objective : **moyenne des scores RMSE des folds** obtenue pour chaque test.

Pur les paramètres du modèle :

- **iterations** : Le nombre d'itérations (ou arbres) à construire lors de l'entraînement du modèle. Plus le nombre d'itérations est élevé,

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)
MSE' indique que le modèle utilise l'erreur quadratique moyenne (Root Mean Squared Error) comme mesure de perte à minimiser.

- **od_wait** : Le nombre d'itérations à attendre (patience) pour déterminer si l'erreur de validation croisée s'améliore. Si aucune amélioration n'est observée pendant ce nombre d'itérations, l'entraînement peut s'arrêter plus tôt pour éviter le surapprentissage.

Si on fixe **od_wait** on peut mettre une grande valeur pour **iterations** car de toute façon le modèle détecte l'overfitting quand il n'y a plus d'amélioration sur la base de validation.

```
# définir l'espace de recherche des hyperparamètres
search_space = list()
search_space.append(Real(1e-3, 0.5, 'log-uniform', name='learning_rate'))
search_space.append(Integer(2, 7, name='depth'))

# définir la fonction utilisée pour évaluer les configurations
@use_named_args(search_space)
def evaluate_model(**params):
    # Choisir une configuration
    print(params)

    # Ce n'est pas grave d'utiliser un si grand nombre d'itération car le paramètre
    # od_wait permet de stopper les itérations quand il n'y a pas d'amélioration
    # de performances sur la base de validation
    default_params = {'iterations' : 10_000,
                      'loss_function' : 'RMSE',
                      'od_wait' : 20,
                      'use_best_model' : True,
                      'verbose' : False,
                      'random_seed' : 0,
                      'cat_features' : cat_features}

    all_params = {**params, **default_params}

    # Data d'entraînement
    data_pool = Pool(X_train, y_train, cat_features=cat_features)

    # Validation croisée
```

Third-party libraries used:
Support
third party
widgets
(widgets
outside
ipywidg
package
to be en
separate
Support
these wi
will be lo
from a C
external
Colab.

from g
output

Support
third par
widgets
remain e
for the d
of the se
To disa
support:

from g
output

ATTIC
notel
cou

```
cv_result = cv(data_pool, all_params, fold_count=4, seed=0, shuffle=True)

# calculer le score RMSE moyen pour l'essaie
result = min(cv_result["test-RMSE-mean"])

return result

# Chercher le min de la fonction objective sur le search_space
result = gp_minimize(evaluate_model, search_space)

# Résultats :
print('Best RMSE: %.3f' % result.fun)
print('Best Parameters: %s' % (result.x))

bestIteration = 1670

Training on fold [2/4]

bestTest = 1.988303314
bestIteration = 911

Training on fold [3/4]

bestTest = 1.452913922
bestIteration = 1255

{'learning_rate': 0.04374967263753425, 'depth': 3}
Training on fold [0/4]

bestTest = 1.751961453
bestIteration = 1241

Training on fold [1/4]

bestTest = 1.158197867
bestIteration = 1694

Training on fold [2/4]

bestTest = 1.948998261
bestIteration = 1054

Training on fold [3/4]

bestTest = 1.558935585
bestIteration = 839
```

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

```
bestTest = 1.644885964
bestIteration = 1660

Training on fold [1/4]

bestTest = 1.234855872
bestIteration = 1458

Training on fold [2/4]

bestTest = 1.851306349
bestIteration = 1162

Training on fold [3/4]

bestTest = 1.479406258
bestIteration = 1253

{'learning_rate': 0.045528511791156225, 'depth': 3}
Training on fold [0/4]

bestTest = 1.739610245
bestIteration = 1164

# Meilleur learning_rate
lr_best = result.x[0]

# Meilleur depth
depth_best = result.x[1]

# Meilleur nombre d'itérations
iter_best = 1130
```

On obtient la meilleure performance (RMSE = 1.46 sur les données de validation) pour learning_rate = 0.05947940568900549 et depth = 3.

▼ 3.3.2. Régularisation

Afin de contrôler davantage la compléxité du modèle et éviter que la variance soit trop élevée, nous optimisons l'hyperparamètre **l2_leaf_reg**.

l2_leaf_reg est un hyperparamètre utilisé dans l'algorithme CatBoost pour appliquer une régularisation L2 (régularisation Ridge) aux poids des feuilles des arbres de décision construits pendant l'entraînement. La régularisation L2 ajoute une pénalité proportionnelle au carré des valeurs des poids des caractéristiques, ce qui a pour effet de limiter leur croissance excessive et de réduire le risque de surapprentissage.

```
# Paramètres du modèle
params = {'iterations' : 10_000,
          'learning_rate' : lr_best,
          'depth' : depth_best,
          'loss_function' : 'RMSE',
          'od_wait' : 20,
          'use_best_model' : True,
          'verbose' : False,
          'random_seed' : 0,
          'cat_features' : cat_features}

# Valeurs de l2_leaf_reg à tester
l2_leaf_reg_values = [1e-3, 2e-3, 5e-3, 7e-3, 1e-2, 2e-2, 5e-2, 7e-2, 0.1, 1, 5, 10]

# Listes pour sauvegarder les résultats des validations croisées
cv_means=[]
cv_vars=[]

# Effectuer une validation croisée par valeur de l2_leaf_reg et stocker les résultats
for l2_leaf_reg in l2_leaf_reg_values:
    # Data
    data_pool = Pool(X_train, y_train, cat_features=cat_features)

    # Validation croisée
    cv_result = cv(data_pool, **params, **{'l2_leaf_reg' : l2_leaf_reg}), fold_count=4, seed=0, shuffle=True)

    # Indice du meilleur score moyen
    index_min = cv_result['test-RMSE-mean'].idxmin()

    # Score moyen obtenu avec la validation croisée
    mean = cv_result.loc[index_min, "test-RMSE-mean"]
    cv_means.append(mean)

    var = cv_result.loc[index_min, "test-RMSE-var"]
    cv_vars.append(var)
```

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

```
Training on fold [0/4]
```

```
bestTest = 1.812443482
bestIteration = 1875
```

```
Training on fold [1/4]
```

```
bestTest = 1.449706393
bestIteration = 1635
```

```
Training on fold [2/4]
```

```
bestTest = 2.32823856
bestIteration = 1119
```

```
Training on fold [3/4]
```

```
bestTest = 2.127508427
bestIteration = 572
```

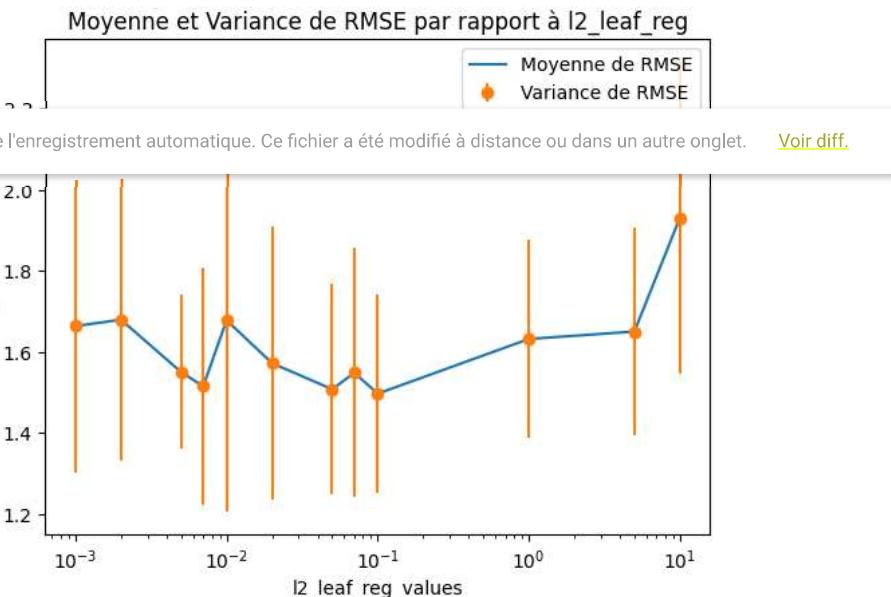
Maintenant, Nous visualisons l'effet de **l2_leaf_reg** sur les performances du modèle.

```
# Echelle log pour l2_leaf_reg
# Tracer la moyenne
plt.semilogx(l2_leaf_reg_values, cv_means, label='Moyenne de RMSE')

# Tracer la variance
plt.errorbar(l2_leaf_reg_values, cv_means, yerr=cv_vars, fmt='o', label='Variance de RMSE')

# Ajouter des titres et des légendes
plt.xlabel('l2_leaf_reg_values')
plt.ylabel('cv_RMSE')
plt.title('Moyenne et Variance de RMSE par rapport à l2_leaf_reg')
plt.legend()

# Afficher le graphique
plt.show()
```



Nous remarquons que la régularisation n'a pas d'effet sur l'amélioration du pouvoir de généralisation du modèle. La régularisation s'avère non nécessaire, d'autant plus que le modèle n'est pas complexe.

3.3.3. Test sur la base de test.

Maintenant, nous mesurons le pouvoir de généralisation du modèle en utilisant la base de test qui n'a pas été utilisée lors de l'entraînement.

```
# Data d'entraînement
data_pool = Pool(X_train, y_train, cat_features=cat_features)

# Paramètres optimaux
params = {'iterations' : iter_best,
          'learning_rate' : lr_best,
          'depth' : depth_best,
```

```
'loss_function' : 'RMSE',
'verbose' : False,
'random_seed' : 0,
'cat_features' : cat_features}

# Entrainement du modèle
cat_model = CatBoostRegressor(**params)
cat_model.fit(X_train,
              y_train,
              plot=True)

# Prédiction sur la base de test
y_pred = cat_model.predict(X_test,
                           prediction_type=None,
                           verbose=True)

# Récupérer le score par rapport aux valeurs réelles
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"RMSE de test est : {rmse_test}")

RMSE de test est : 1.3795619905003735
```

Nous remarquons que RMSE du test est proche de RMSE de validation (même inférieur!). Ce qui veut dire que le modèle a un bon pouvoir de généralisation.

Les hyperparamètres sont alors validés !

▼ 3.4 Modèle final

Maintenant, pour le modèle final, on entraîne le modèle sur la base d'entraînement entière en utilisant les hyperparamètres optimaux, tout en gardant un œil sur la variance et le biais du modèle en utilisant la base de test cette fois comme base de validation pour le modèle. Ainsi, ceci permettra d'entraîner le modèle sur plus de données tout en évitant l'overfitting.

```
# Data d'entraînement
data_pool = Pool(X_train, y_train, cat_features=cat_features)

# Data de validation
eval_pool = Pool(X_test, y_test, cat_features=cat_features)

# Paramètres optimaux
Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. Voir diff.

'depth' : depth_best,
'loss_function' : 'RMSE',
'use_best_model' : True,
'od_wait' : 20,
'verbose' : False,
'random_seed' : 0,
'cat_features' : cat_features}

# Entraînement du modèle
cat_model_final = CatBoostRegressor(**params)
cat_model_final.fit(X_train,
                    y_train,
                    plot=True,
                    eval_set=eval_pool,
                    plot_file='/content/drive/MyDrive/Study case/plot_train.html',
                    verbose_eval=True)
```



On obtient pour 845 itérations :

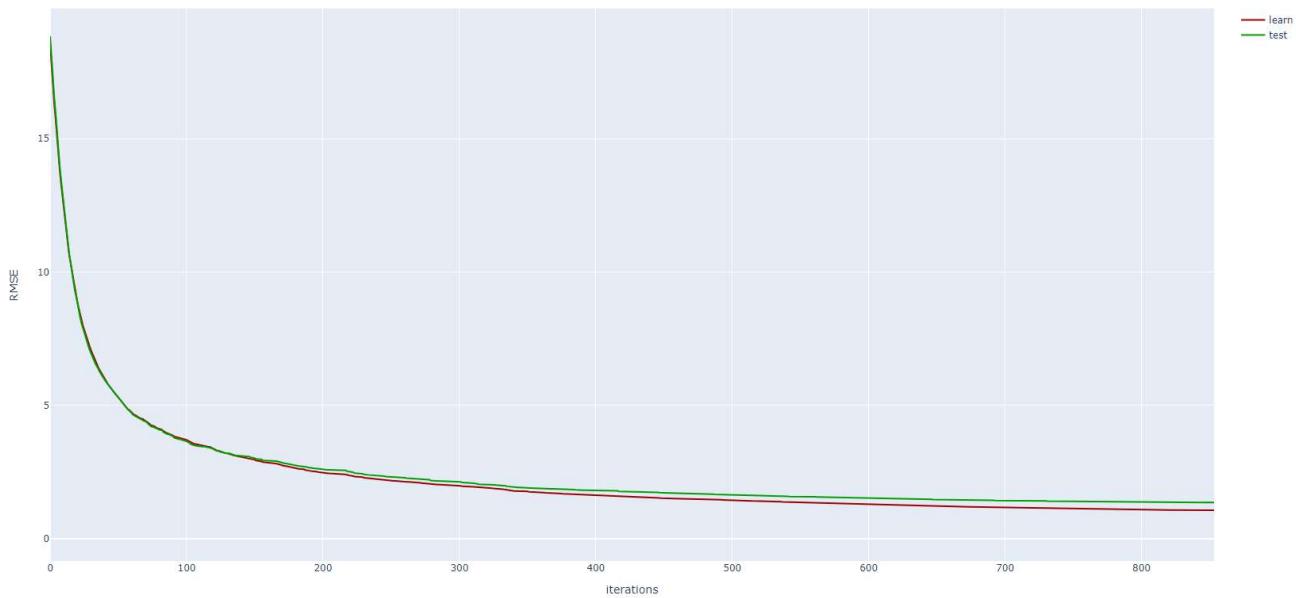
- RMSE d'entraînement : 1.07
- RMSE de validation : 1.37

On obtient pour 845 itérations :

- RMSE d'entraînement : 1.07
- RMSE de validation : 1.37

[Courbe d'entraînement/validation](#)

Training plots : RMSE

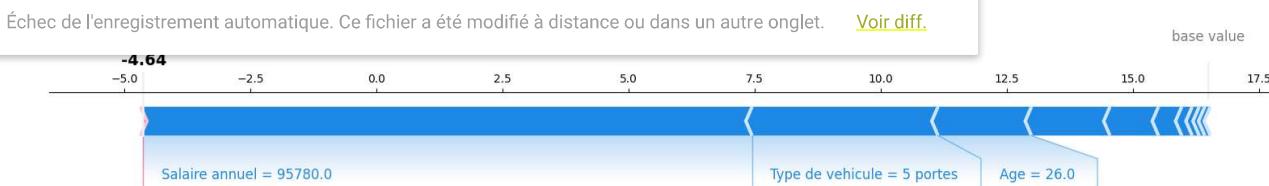


▼ 3.5. Explicabilité et Importance des features

On utilise les shap values pour regarder l'influence des variables sur la cible.

```
# Récupérer les valeurs shap
explainer = shap.TreeExplainer(cat_model_final)
shap_values = explainer.shap_values(X_train)

# Visualiser la prédiction sur la première ligne
shap.force_plot(explainer.expected_value, shap_values[0,:], X_train.iloc[0,:],
                 matplotlib=True)
```

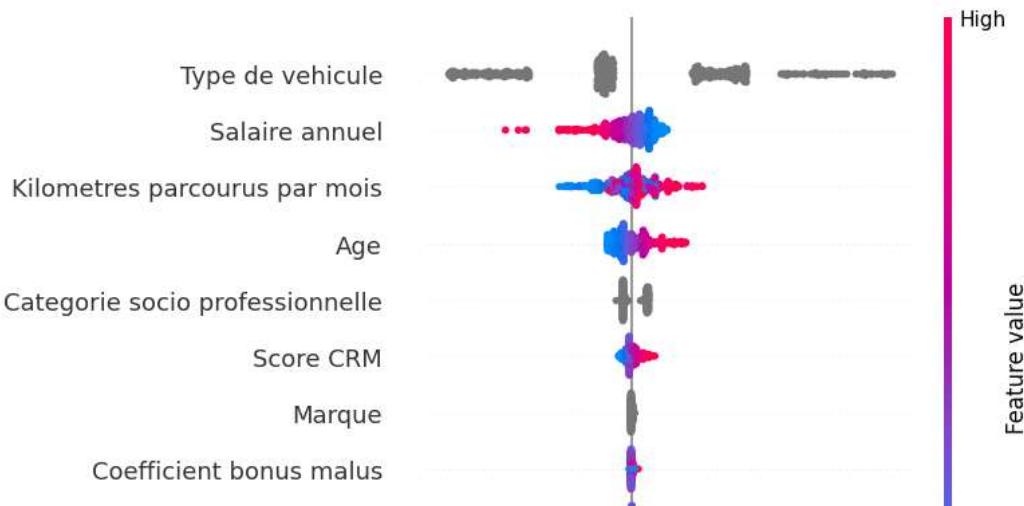


On peut regarder quels sont les variables qui ont contribué à prédire la valeur -4.45 pour cette ligne. La variable **Salaire annuel** suivi du **Type de véhicule** puis **Age** contribuent à pousser la valeur prédictive aux valeurs inférieures.

```
# Effet de toutes les variables
shap.summary_plot(shap_values, X_train)
```

```
/usr/local/lib/python3.10/dist-packages/shap/plots/_beeswarm.py:699: UserWarning:
```

No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored



Ceci valide la remarque qu'on a faite pendant l'analyse descriptive : que le Type de véhicule est un facteur discriminant pour déterminer la valeur du bénéfice.

▼ 4. Prédiction

-20 0 20 40

▼ 4.1 Traitement des données de soumission

La data de soumission doit subir les mêmes changements effectués sur la data d'entraînement pour qu'elle soit acceptable par le modèle

```
submission.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 0 to 299
```

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

		index	300	non-null	int64
0	Age	296	non-null	float64	
1	Prime mensuelle	299	non-null	float64	
2	Catégorie socio professionnelle	300	non-null	object	
3	Kilometres parcourus par mois	300	non-null	int64	
4	Coefficient bonus malus	300	non-null	int64	
5	Type de véhicule	300	non-null	object	
7	Score CRM	300	non-null	int64	
8	Niveau de vie	300	non-null	int64	
9	Marque	284	non-null	object	
10	Salaire annuel	300	non-null	int64	
11	Score credit	300	non-null	int64	
12	Cout entretien annuel	300	non-null	int64	

dtypes: float64(2), int64(8), object(3)
memory usage: 30.6+ KB

▼ 4.1.1. Colonnes supprimées

```
submission.drop(columns = ['Niveau de vie'], inplace = True)
```

▼ 4.1.2. Valeurs manquantes

Variable Marque

Nous renseignons les champs vides de la variable Marque comme on l'a fait pour la base d'entraînement

```
# le nombre de champs vides dans la colonne Marque
num_missing = submission['Marque'].isnull().sum()

categories = train['Marque'].value_counts().to_dict()
```

```
N = sum(list(categories.values()))

cat_prob = {key : value/N for key, value in categories.items()}

# Générez des échantillons aléatoires pour remplir les champs vides
random_samples = np.random.choice(list(categories.keys()), size=num_missing, p=list(cat_prob.values()))

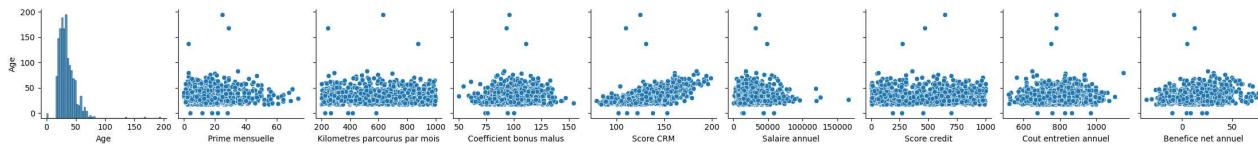
# Remplacer les champs vides par les échantillons aléatoires générés
submission.loc[submission['Marque'].isnull(), 'Marque'] = random_samples
```

Variable Age

Nous regardons la corrélation de la variable **Age** avec les autres variables.

```
# Pour les variables numériques

sns.pairplot(train[[col for col in train.columns if col != 'index']], y_vars = ["Age"])
plt.show()
```



```
# Pour les variables catégorielles
```

```
cat_columns = train.select_dtypes(include=['object']).columns.tolist()
fig = make_subplots(rows=1,
                     cols=len(cat_columns),
                     subplot_titles=(cat_columns[0], cat_columns[1], cat_columns[2]),
                     )

col1 = go.Box(x = train[cat_columns[0]], y=train["Age"], name =cat_columns[0])
col2 = go.Box(x = train[cat_columns[1]], y=train["Age"], name =cat_columns[1])
```

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)

```
+fig.add_trace(col1, row = 1, col = 1)
fig.add_trace(col2, row = 1, col = 2)
fig.add_trace(col3, row = 1, col = 3)

# Limiter l'axe y à 85 pour chaque sous-graphique
fig.update_yaxes(range=[0, 85], row=1, col=1)
fig.update_yaxes(range=[0, 85], row=1, col=2)
fig.update_yaxes(range=[0, 85], row=1, col=3)

fig.update_layout(showlegend = False)
fig.show()
```

On choisit de générer un vecteur qui suit la loi de génération de la variable Age

```
# Générer un vecteur de 4 âge en imitant la loi dans la base d'entraînement
size = submission['Age'].isnull().sum()
np.random.seed(0)
mean = np.mean(train["Age"])
std = np.std(train["Age"])
generated_age = [int(elt) for elt in list(np.random.normal(mean, std, size))]
print("Vecteur age généré :", generated_age)

Vecteur age généré : [59, 40, 48, 66]
```

```
# Reseigner les champs vides
submission.loc[submission['Age'].isnull(), 'Age'] = generated_age
```

Variable Prime mensuelle

```
    0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100
```

On fait de même pour la prime mensuelle

```
# Générer un vecteur prime mensuelle en imitant la loi dans la base d'entraînement
size = submission['Prime mensuelle'].isnull().sum()
np.random.seed(0)
mean = np.mean(train["Prime mensuelle"])
std = np.std(train["Prime mensuelle"])
generated_prime_mensuelle = list(np.random.normal(mean, std, size))
print("Vecteur prime mensuelle généré :", generated_prime_mensuelle)

Vecteur prime mensuelle généré : [42.088501063860015]
```

```
submission.loc[submission['Prime mensuelle'].isnull(), 'Prime mensuelle'] = generated_prime_mensuelle
```

▼ 4.2. Comparaison de data d'entraînement et data de prédiction

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#) que les instances des données catégorielles sont les mêmes). On procéde à une comparaison des distributions, au moins pour les 4 ou 5 variables plus importantes pour le modèle.

```
def compare_distributions(train_df, submission_df, column, is_categorical):

    if not is_categorical :
        fig = make_subplots(rows=1,
                            cols=2,
                            subplot_titles=("Data d'entraînement", "Data de prédiction"),
                            x_title= column,
                            y_title=column,
                            shared_yaxes=True)
        train_dist = go.Box(y = train_df[column], name = '')
        pred_dist = go.Box(y = submission_df[column], name = '')

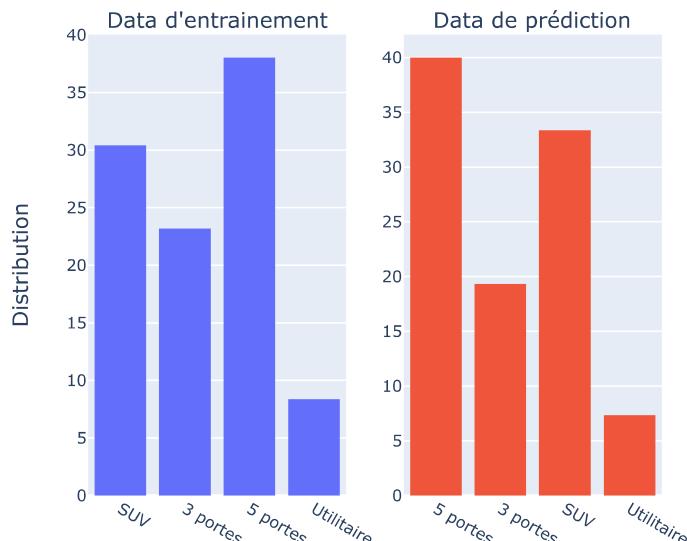
    else :
        fig = make_subplots(rows=1,
                            cols=2,
                            subplot_titles=("Data d'entraînement", "Data de prédiction"),
                            x_title= column,
                            y_title='Distribution')
        train_dist = go.Histogram(x = train_df[column], histnorm='percent')
        pred_dist = go.Histogram(x = submission_df[column], histnorm='percent')

    fig.add_trace(train_dist, row = 1, col = 1)
    fig.add_trace(pred_dist, row = 1, col = 2)

    fig.update_layout(showlegend=False, title_text=f"Distribution de {column}")
    fig.show()

compare_distributions(train, submission, "Type de véhicules", True)
```

Distribution de Type de véhicule



```
compare_distributions(train, submission, "Salaire annuel", False)
```

Distribution de Salaire annuel



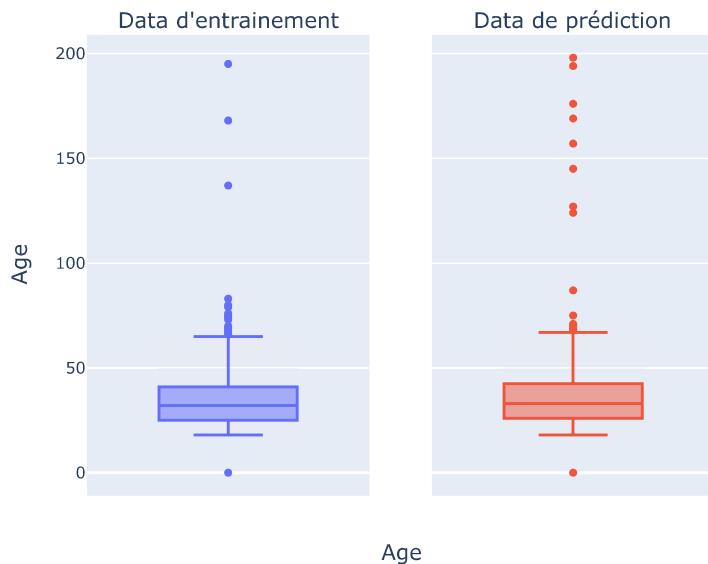
```
compare_distributions(train, submission, "Kilometres parcourus par mois", False)
```

Distribution de Kilometres parcourus par mois

Data d'entraînement Data de prédition

```
compare_distributions(train, submission, "Age", False)
```

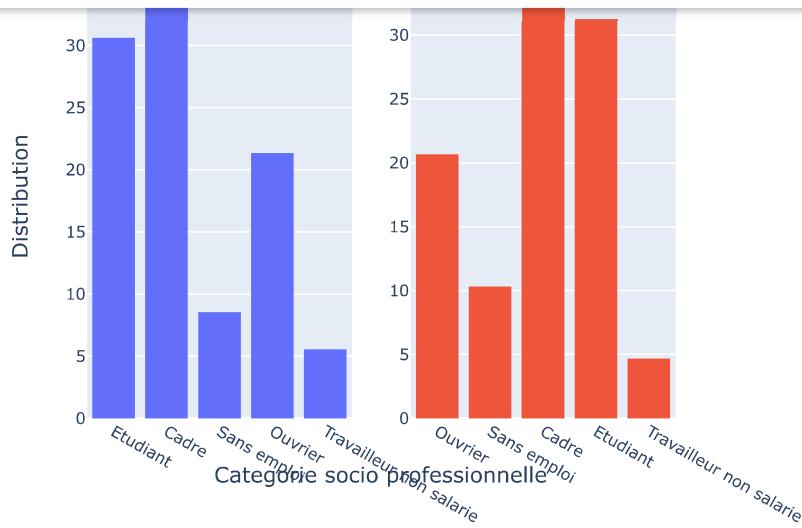
Distribution de Age



```
compare_distributions(train, submission, "Categorie socio professionnelle", True)
```

Distribution de Categorie socio professionnelle

Échec de l'enregistrement automatique. Ce fichier a été modifié à distance ou dans un autre onglet. [Voir diff.](#)



On remarque que la distribution est la même pour les variables les plus importantes.

4.3. Prédiction et soumission

```
# données de prédiction
X_pred = submission.drop(columns = ['index'])
y_pred = cat_model_final.predict(X_pred)
```

```
submission["Prediction_benefice"] = y_pred
```