

PROJET DATA SCIENCE

LES ASSURANCES

Le jeu de données que nous possédons répertorie en fonction de différents paramètres influençant le clv ("customer lifetime value"). Après nos recherches on a vu que celui-ci se définit comme une mesure commerciale qui estime le taux total d'achats répétés prévisibles d'un client au cours de la durée de vie de son temps avec la marque. Plus le CLV (valeur à vie) du client est élevé, plus il est considéré comme précieux dans l'entreprise. La problématique qui a été donnée avec le jeu de données est celle-ci : Quelles caractéristiques impactent le clv? Comment prévoir sa valeur?

Nous réalisons d'abord ci-dessous l'importation de toutes les bibliothèques nécessaires

```
In [1]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sb
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import os
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_regression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
In [2]: df=pd.read_csv('VehicleInsuranceData.csv', sep=',')
```

```
In [3]: df
```

	Unnamed: 0	clv	Response	Coverage	Education	EmploymentStatus	Gender	Income
0	1	2763.519279	No	Basic	Bachelor	Employed	F	56274
1	2	6979.535903	No	Extended	Bachelor	Unemployed	F	0
2	3	12887.431650	No	Premium	Bachelor	Employed	F	48767
3	4	7645.861827	No	Basic	Bachelor	Unemployed	M	0
4	5	2813.692575	No	Basic	Bachelor	Employed	M	43836
...
8625	9129	4100.398533	No	Premium	College	Employed	F	47761

Unnamed: 0	clv	Response	Coverage	Education	EmploymentStatus	Gender	Income
8626	9131	3096.511217	Yes	Extended	College	Employed	F 21604
8627	9132	8163.890428	No	Extended	Bachelor	Unemployed	M 0
8628	9133	7524.442436	No	Extended	College	Employed	M 21941
8629	9134	2611.836866	No	Extended	College	Unemployed	M 0

8630 rows × 22 columns

Description des données

Nous allons commencer par décrire les données numériques pour ensuite décrire les données de chaîne de caractères. En effet le type de données est important lorsque l'on cherche à effectuer des analyses. Nous allons donc par la suite transformer ces données en format numérique d'où l'intérêt de connaître tous les différents champs des colonnes.

Description des colonnes et de leurs valeurs

La colonne suivante représente le risque lié à l'assurance. Cette colonne représente l'une des colonnes sur laquelle notre étude va se porter. On va chercher à savoir quels paramètres influent sur le risque d'une assurance. On voit alors qu'il y a deux types de risque : le cas où l'assurance ne représente pas de danger pour l'entreprise et le cas où l'assurance représente un risque pour l'entreprise.

Colonne représentant le type d'assurance

```
In [4]: df['Coverage'].unique()
Out[4]: array(['Basic', 'Extended', 'Premium'], dtype=object)
```

Colonne représentant le niveau d'études

```
In [5]: df['Education'].unique()
Out[5]: array(['Bachelor', 'College', 'Master', 'High School or Below', 'Doctor'],
              dtype=object)
```

Colonne représentant le statut d'emploi

```
In [6]: df['EmploymentStatus'].unique()
Out[6]: array(['Employed', 'Unemployed', 'Medical Leave', 'Disabled', 'Retired'],
              dtype=object)
```

Colonne représentant le sexe

```
In [7]: df['Gender'].unique()
```

```
Out[7]: array(['F', 'M'], dtype=object)
```

Colonne représentant le status de l'individu(marié, célibataire...)

```
In [8]: df['Marital.Status'].unique()
```

```
Out[8]: array(['Married', 'Single', 'Divorced'], dtype=object)
```

Colonne représentant l'environnement dans lequel la conduite de la voiture est adaptée

```
In [9]: df['Location.Code'].unique()
```

```
Out[9]: array(['Suburban', 'Rural', 'Urban'], dtype=object)
```

colonne indiquant le type de politique de la voiture

```
In [10]: df['Policy.Type'].unique()
```

```
Out[10]: array(['Corporate Auto', 'Personal Auto', 'Special Auto'], dtype=object)
```

colonne représentant le type d'offre

```
In [11]: df['Renew.Offer.Type'].unique()
```

```
Out[11]: array(['Offer1', 'Offer3', 'Offer2', 'Offer4'], dtype=object)
```

colonne représentant la méthode de vente

```
In [12]: df['Sales.Channel'].unique()
```

```
Out[12]: array(['Agent', 'Call Center', 'Web', 'Branch'], dtype=object)
```

```
In [13]: df['Vehicle.Class'].unique()
```

```
Out[13]: array(['Two-Door Car', 'Four-Door Car', 'SUV', 'Luxury SUV', 'Sports Car', 'Luxury Car'], dtype=object)
```

```
In [14]: df['Vehicle.Size'].unique()
```

```
Out[14]: array(['Medsize', 'Small', 'Large'], dtype=object)
```

Ici on affiche les principales caractéristiques de nos tables(myennes, valeurs, max...) pour en déduire certaines dependances et certains paramètres pour la suite

```
In [53]: df.describe()
```

	Unnamed: 0	clv	Income	Monthly.Premium.Auto	Months.Since.Last.Claim	Mc
count	8630.000000	8630.000000	8630.000000	8630.000000	8630.000000	8630.000000
mean	4560.576825	6725.281515	37586.405794	91.677057	15.084473	
std	2637.448711	3934.876313	30453.838594	32.648209	10.048979	
min	1.000000	1898.007675	0.000000	61.000000	0.000000	
25%	2280.250000	3858.177183	0.000000	68.000000	6.000000	
50%	4547.500000	5569.220419	33817.000000	81.000000	14.000000	
75%	6852.750000	8456.408530	62250.750000	108.000000	23.000000	
max	9134.000000	21235.445570	99981.000000	298.000000	35.000000	



In [15]: `df.index`

Out[15]: `RangeIndex(start=0, stop=8630, step=1)`

In [16]: `df.columns`

Out[16]: `Index(['Unnamed: 0', 'clv', 'Response', 'Coverage', 'Education', 'EmploymentStatus', 'Gender', 'Income', 'Location.Code', 'Marital.Status', 'Monthly.Premium.Auto', 'Months.Since.Last.Claim', 'Months.Since.Policy.Inception', 'Number.of.Open.Complaints', 'Number.of.Policies', 'Policy.Type', 'Policy', 'Renew.Offer.Type', 'Sales.Channel', 'Total.Claim.Amount', 'Vehicle.Class', 'Vehicle.Size'], dtype='object')`

Recherche des paramètres ayant une influence sur notre target (clv)

Ici on va utiliser une certaine méthode nous permettant de déterminer quels paramètres ont une influence sur nos données afin de réaliser des méthodes de machine learning pertinentes

En utilisant la matrice de corrélation

La matrice de corrélation

```
In [75]: corr = df.corr()
corrMat = plt.matshow(corr, fignum = 1)
plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
plt.yticks(range(len(corr.columns)), corr.columns)
sb.heatmap(corr, annot=True)
plt.show()
```



On se place sur la ligne de clv et on peut voir quels paramètres ont une influence sur notre target :

Monthly.Premium.Auto

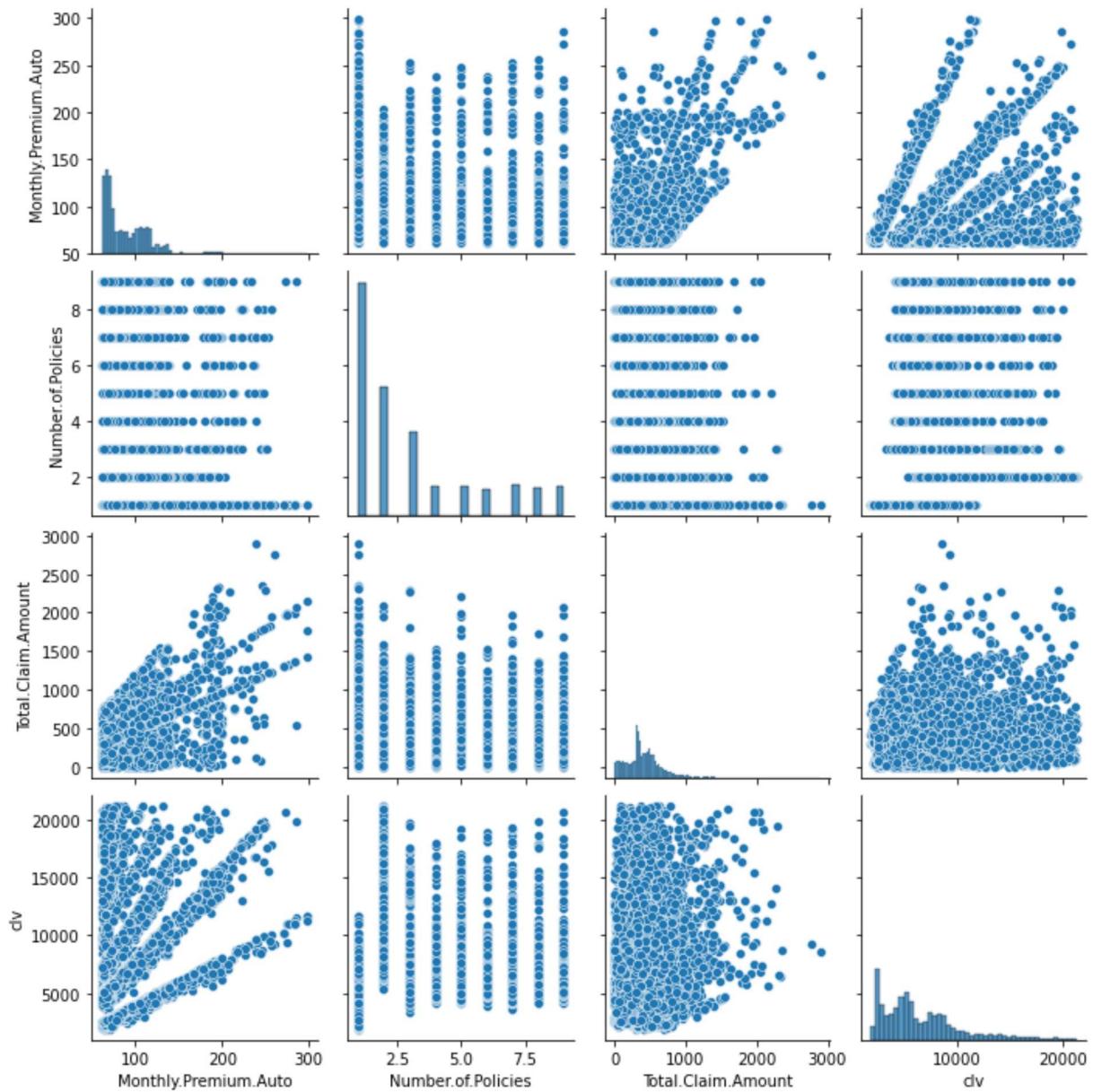
Number.of.Policies

Total.Claim.Amount

On peut alors tracer les courbes qui nous intéressent grâce aux différentes fonctions graphiques qui ont été introduites lors des premiers tps:

Dans un premier temps avec les valeurs originales de clv (qui prend une grande étendue de valeurs):

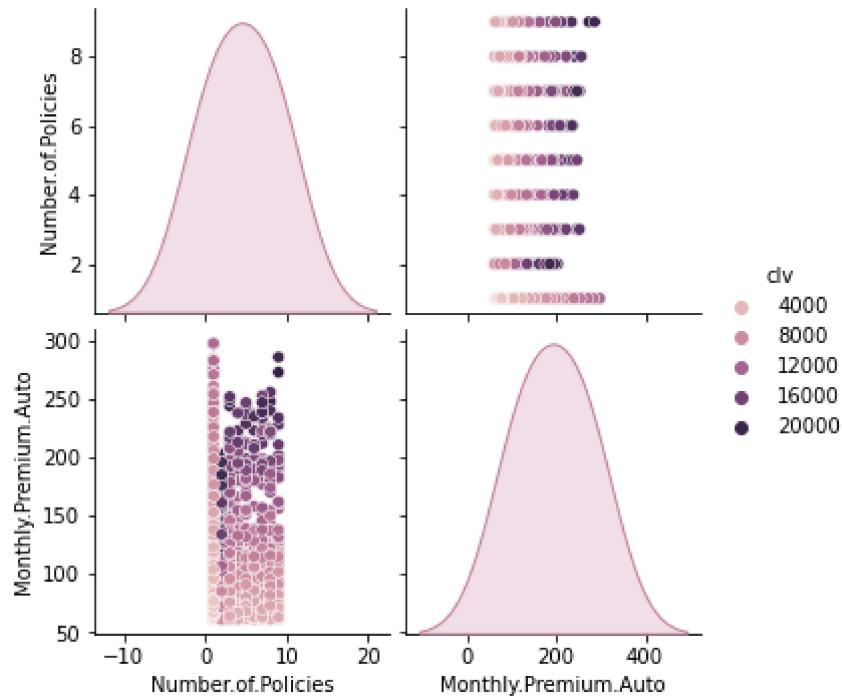
```
In [128...]: sb.pairplot(data=df, vars=['Monthly.Premium.Auto', 'Number.of.Policies', 'Total.Claim.A
Out[128...]: <seaborn.axisgrid.PairGrid at 0x1bc52b3bdc0>
```



```
In [17]: datafram=df.sample(n=3000)
```

```
In [16]: sb.pairplot(data=df,vars=[ 'Number.of.Policies','Monthly.Premium.Auto'],hue='clv')
```

```
Out[16]: <seaborn.axisgrid.PairGrid at 0x1bc4c112430>
```



```
In [18]: df['Total.Claim.Amount']
```

```
Out[18]: 0      384.811147
1      1131.464935
2      566.472247
3      529.881344
4      138.130879
...
8625    541.282007
8626    379.200000
8627    790.784983
8628    691.200000
8629    369.600000
Name: Total.Claim.Amount, Length: 8630, dtype: float64
```

Créons un dataset(df2) composé des colonnes qui nous intéressent pour lui appliquer nos méthodes de machine learning par la suite

```
In [19]: columns = ['Monthly.Premium.Auto', 'Number.of.Policies', 'Total.Claim.Amount']
df2 = df[['Monthly.Premium.Auto', 'Number.of.Policies', 'Total.Claim.Amount', 'clv']]
```

```
In [20]: df2
```

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount	clv
0	69	1	384.811147	2763.519279
1	94	8	1131.464935	6979.535903
2	108	2	566.472247	12887.431650
3	106	7	529.881344	7645.861827
4	73	1	138.130879	2813.692575
...
8625	104	1	541.282007	4100.398533

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount	clv
8626	79	1	379.200000	3096.511217
8627	85	2	790.784983	8163.890428
8628	96	3	691.200000	7524.442436
8629	77	1	369.600000	2611.836866

8630 rows × 4 columns

Le travail de prétraitement de nos données n'est pas encore terminé, il faut maintenant normaliser les données de certaines colonnes notamment la colonne Total.Claim.Amount qui prend de bien trop grandes valeurs comparé aux autres colonnes :

Normalisation des données

In [21]:

```
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer

scaler = Normalizer().fit(df2)
normalizerData = scaler.transform(df2)
normalizerData = pd.DataFrame(normalizerData, index = df2.index, columns = df2.columns)

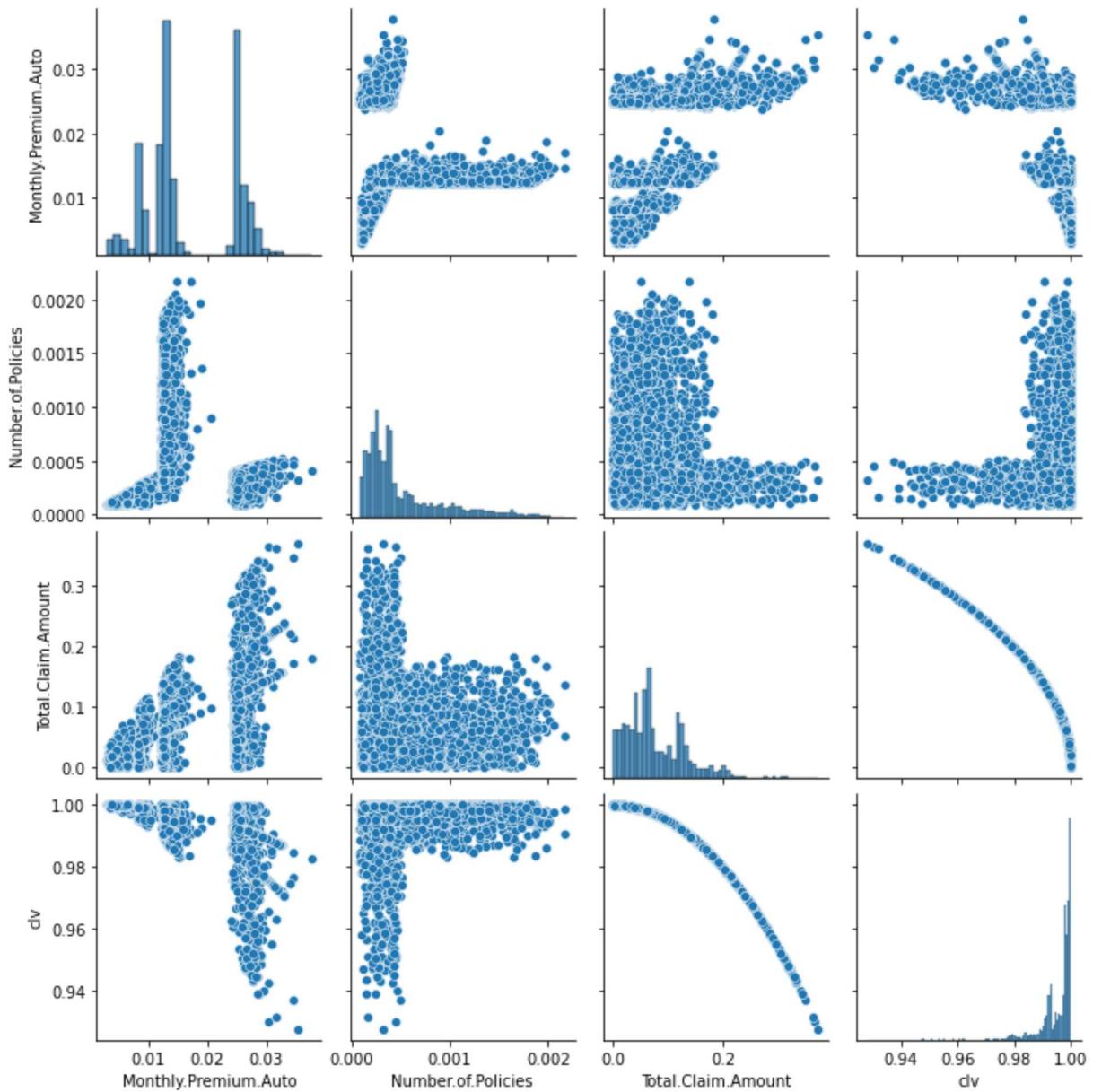
df2=normalizerData
```

In [152...]

```
sb.pairplot(data=df2, vars=['Monthly.Premium.Auto', 'Number.of.Policies', 'Total.Claim.
```

Out[152...]

```
<seaborn.axisgrid.PairGrid at 0x1bc023ffca0>
```



Application des différentes méthodes de machine learning

1) La régression linéaire

Il paraît intéressant de tracer clv en fonction du total.claim.amount. On voit en effet que ces deux variables sont presque linéairement liées sur un intervalle. La régression linéaire va alors nous permettre de vérifier si la dépendance peut être approchée par une dépendance linéaire et par la suite si des prédictions vont être possibles.

In [22]:

df2

Out[22]:

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount	clv
0	0.024722	0.000358	0.137874	0.990141
1	0.013293	0.001131	0.160008	0.987025
2	0.008372	0.000155	0.043911	0.999000
3	0.013829	0.000913	0.069131	0.997511

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount	clv
4	0.025905	0.000355	0.049017	0.998462
...
8625	0.025137	0.000242	0.130830	0.991086
8626	0.025315	0.000320	0.121513	0.992267
8627	0.010363	0.000244	0.096407	0.995288
8628	0.012704	0.000397	0.091468	0.995727
8629	0.029178	0.000379	0.140054	0.989714

8630 rows × 4 columns

```
In [23]: array = df2.values.tolist()
```

```
In [24]: array[0][2]
```

```
Out[24]: 0.13787395962155485
```

```
In [25]: #on transforme la série en vecteur
# on commence par la target (clv)
array = df2.values.tolist()
liste_clv=[]
for i in range(5000):
    liste_clv.append (array[i][3])
liste_clv=np.array(liste_clv)
matrice_clv=liste_clv.reshape(liste_clv.shape[0], 1)
```

```
In [26]: #tca(total claim amount)
liste_tca=[]
for i in range(5000):
    liste_tca.append (array[i][2])
liste_tca=np.array(liste_tca)
matrice_tca=liste_tca.reshape(liste_tca.shape[0], 1)
```

```
In [27]: #on initialise theta avec des valeurs par défaut
#np.random.seed(0)
theta = np.random.randn(2,1)
theta
```

```
Out[27]: array([[-1.00102667],
 [ 1.32330913]])
```

```
In [28]: #Matrice X ( le clv)
x=matrice_clv
X=np.hstack((matrice_clv,np.ones(matrice_clv.shape)))
X.shape
```

```
Out[28]: (5000, 2)
```

In [29]:

```
#Matrice Y ( Les total claim amount)
y=matrice_tca
Y=np.hstack((matrice_tca,np.ones(matrice_tca.shape)))

print(x.shape)
print(y.shape)

print(X.shape)
```

```
(5000, 1)
(5000, 1)
(5000, 2)
```

1.1)Modèle linéaire

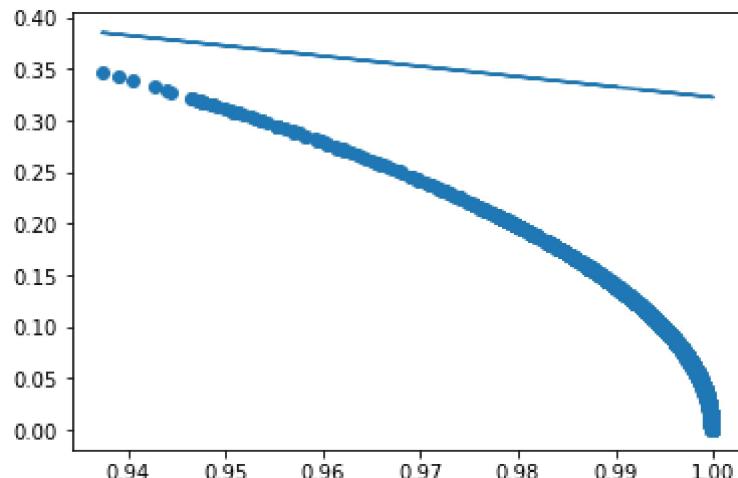
In [30]:

```
def model(X, theta):
    #produit matriciel
    return X.dot(theta)
```

In [31]:

```
plt.plot(x, model(X,theta))
plt.scatter(x,y)
```

Out[31]:



(clv en abscisse et tca en ordonnée)

On rappel que l'on a initialisé theta avec des valeurs aléatoires ceci explique pourquoi le modèle n'est pas encore adaptés l'enjeux de la suite de cette partie est de prendre des valeurs adaptées.

définissons une fonction cout erreur qui nous permettra d'évaluer notre modèle par la suite

In [32]:

```
def error_function(x,y,theta):
    m=len(y)
    return 1/(2*m) * np.sum((model(X,theta)-y)**2)
```

In [33]:

```
error_function(X,y,theta)
```

Out[33]: 0.03234935969193539

En voyant la valeur prise par l'erreur on peut se dire que notre modèle est déjà performant sauf qu'il ne faut pas oublier qu'après la normalisation nous nous retrouvons avec des valeurs très petites et une telle erreur sur nos données est en réalité beaucoup trop grande.

1.2) Algorithme de la descente de gradient

In [34]:

```
def grad(X,y,theta):
    m=len(y)
    return (1/m)* X.T.dot(model(X,theta)) - y
```

In [35]:

```
grad(X,y,theta)
```

Out[35]:

```
array([[0.24848856],
       [0.24937125]])
```

In [36]:

```
def gradient_descent(X,y,theta,learning_rate,n_itterations):
    for i in range(0,n_itterations):
        theta = theta - learning_rate * grad(X,y,theta)
    return theta
```

1.3) Machine learning

In [37]:

```
theta_learning=gradient_descent(X,y,theta,1,50000)
```

In [38]:

```
#on affiche les paramètres de notre modèle
theta_learning
```

Out[38]:

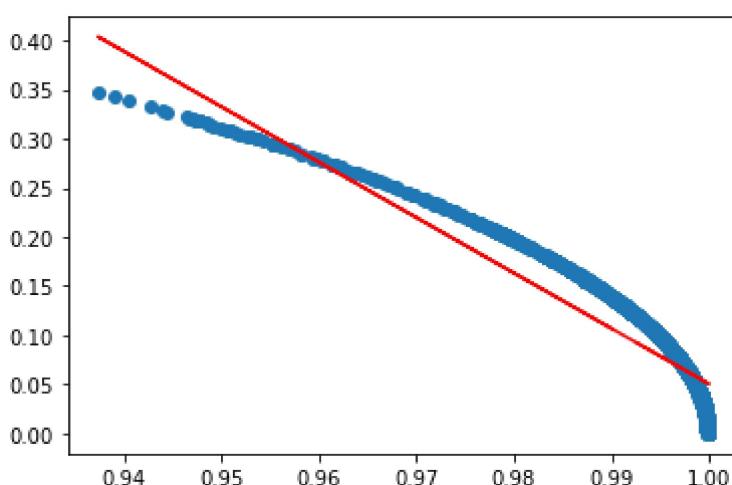
```
array([-5.64218659],
      [ 5.69262861])
```

In [39]:

```
prediction= model(X,theta_learning)
plt.scatter(x,y)
plt.plot(x,prediction, c='r')
```

Out[39]:

```
[<matplotlib.lines.Line2D at 0x2070b8948e0>]
```



In [40]:

```
error_function(X,y,theta_learning)
```

```
Out[40]: 0.0003002591927779098
```

CONCLUSION PARTIELLE (la régression linéaire) :

On voit alors que nous avons réussi à trouver une méthode de machine learning plutôt pertinante que l'on va pouvoir proposer à notre client.

Notre client voulait savoir quels paramètres ont une influence sur le clv de ses clients.

Grace à notre modèle, il va pouvoir prédire en fonction du total claim amount la valeur du clv d'un client.

Méthode 2 : Arbre de décision

Pour appliquer cette méthode de machine learning on va transformer les valeurs des clv en deux classes grâce à la moyenne de cette colonne:

- Les lignes où la valeur est inférieure à la moyenne seront mises à 0
- Les lignes où la valeur est supérieure à la moyenne seront mises à 1

Voici ci-dessous la fonction permettant de réaliser l'opération décrite ci-dessus(classes 0 ou 1) qu'il est nécessaire d'appliquer que pour certaines méthodes

```
In [41]: moyenne=df2['clv'].mean()
j=0
i=int(0)

for i in range (8630) :
    #print(i)
    #print(df2.iloc[1,3])
    if df2.iloc[j,3]< moyenne:
        df2.iloc[j,3]=0
    else:
        df2.iloc[j,3]=1
    j=j+1
df2
```

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount	clv
0	0.024722	0.000358	0.137874	0.0
1	0.013293	0.001131	0.160008	0.0
2	0.008372	0.000155	0.043911	1.0
3	0.013829	0.000913	0.069131	1.0
4	0.025905	0.000355	0.049017	1.0
...

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount	clv
8625	0.025137	0.000242	0.130830	0.0
8626	0.025315	0.000320	0.121513	0.0
8627	0.010363	0.000244	0.096407	1.0
8628	0.012704	0.000397	0.091468	1.0
8629	0.029178	0.000379	0.140054	0.0

8630 rows × 4 columns

In [42]:

```
resultat = df2.groupby('clv').nunique()
resultat
```

Out[42]:

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount
clv			
0.0	2478	2478	2478
1.0	5127	5127	5127

In [43]:

```
X = df2.iloc[:,0:3] #Les caractéristiques
y = df2.iloc[:, 3] #Les résultats (classes)
```

In [44]:

X

Out[44]:

	Monthly.Premium.Auto	Number.of.Policies	Total.Claim.Amount
0	0.024722	0.000358	0.137874
1	0.013293	0.001131	0.160008
2	0.008372	0.000155	0.043911
3	0.013829	0.000913	0.069131
4	0.025905	0.000355	0.049017
...
8625	0.025137	0.000242	0.130830
8626	0.025315	0.000320	0.121513
8627	0.010363	0.000244	0.096407
8628	0.012704	0.000397	0.091468
8629	0.029178	0.000379	0.140054

8630 rows × 3 columns

On sépare nos données en un ensemble de test et un ensemble d'entraînement

In [45]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
```

On va adapter la fonction suivante à notre étude qui permet de donner le max depth le plus adapté

In [46]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

def dtree_grid_search(X,y,nfolds):
    #create a dictionary of all values we want to test
    param_grid = { 'criterion':['gini','entropy'], 'max_depth': np.arange(3, 15)}
    # decision tree model
    dtree_model=DecisionTreeClassifier()
    #use gridsearch to test all values
    dtree_gscv = GridSearchCV(dtree_model, param_grid, cv=nfolds)
    #fit model to data
    dtree_gscv.fit(X, y)
    return dtree_gscv.best_params_
```

In [47]:

```
dtree_grid_search(X_train,y_train,5)
```

Out[47]:

```
{'criterion': 'gini', 'max_depth': 4}
```

On va alors par la suite construire un arbre de profondeur maximale 3

On construit notre arbre de décision

In [48]:

```
tree_clf=DecisionTreeClassifier(max_depth=4)
tree_clf.fit(X_train,y_train)
```

Out[48]:

```
▼      DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4)
```

Evaluation

In [50]:

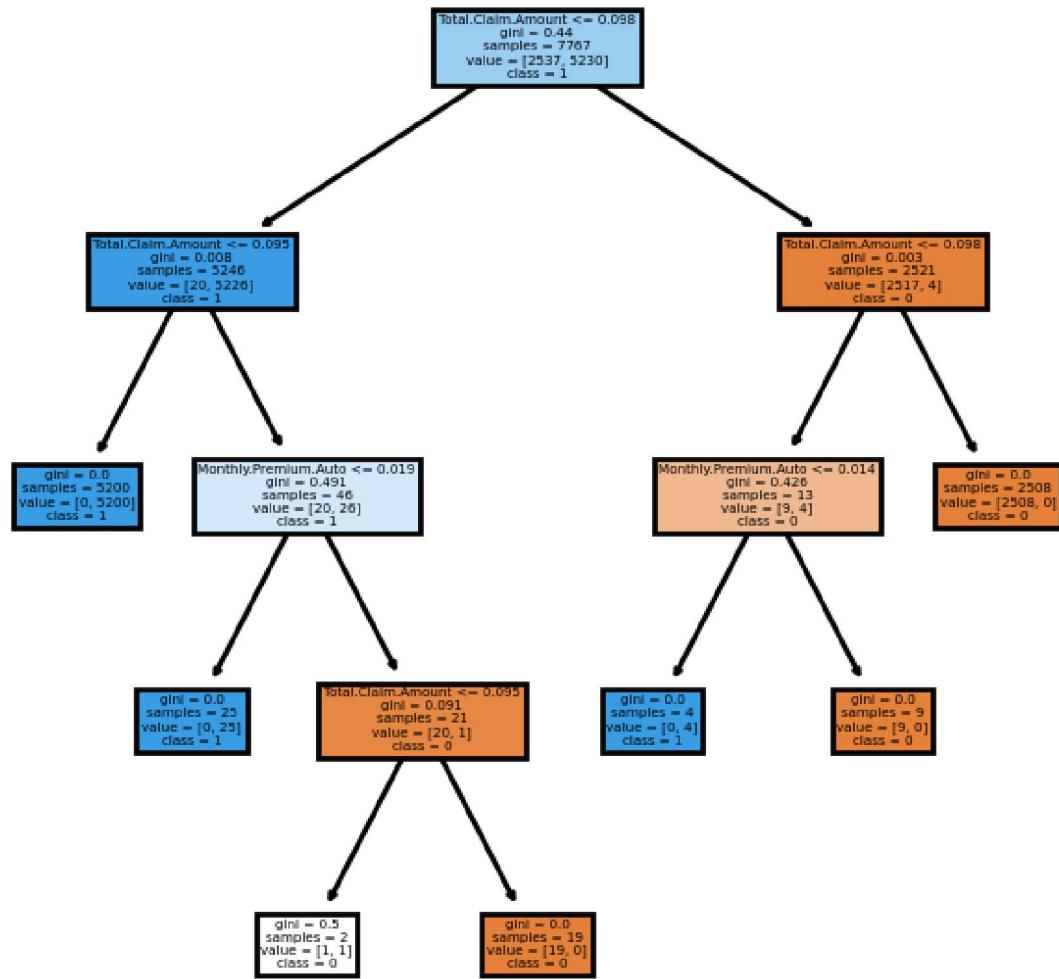
```
y_pred = tree_clf.predict(X_test)
y_score = tree_clf.score(X_test, y_test)
print('Accuracy: ', y_score)
```

Accuracy: 0.9988412514484357

On affiche notre arbre :

In [51]:

```
import matplotlib.pyplot as plt
from sklearn import tree
fn=['Monthly.Premium.Auto','Number.of.Policies','Total.Claim.Amount']
cn=['0', '1']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=200)
tree.plot_tree(tree_clf,
              feature_names = fn,
              class_names=cn,
              filled = True);
fig.savefig('imagename.png')
```



Interprétation du score de notre modèle :

Notre modèle ne possédant qu'une profondeur de 4, il est facile d'examiner directement l'arbre pour évaluer le critère de Gini, qui fournit une indication de l'impureté de nos données par rapport à leurs classifications.

En effet, en examinant les extrémités de nos arbres, on constate que de nombreux facteurs de Gini atteignent la valeur 0, tandis que pour d'autres, cette valeur est proche de 0. Cela démontre la fiabilité de notre modèle, rendue possible grâce à une étude préalable de la profondeur optimale de l'arbre.

Nous sommes donc en mesure de proposer ce modèle au client.

CONCLUSION PARTIELLE 2 (arbre de décision) :

Nous avons précédemment appliqué une méthode de régression linéaire. Cette méthode se basait sur un paramètre pour déduire la CLV (Customer Lifetime Value). Avec cette méthode, notre client sera en mesure d'utiliser différents paramètres à sa disposition pour suivre les différentes branches de l'arbre et ainsi déterminer la classe de son client.

Cette méthode peut être proposée au client en complément du schéma, car la profondeur de l'arbre offre une excellente lisibilité. De plus, nous avons obtenu d'excellents résultats.

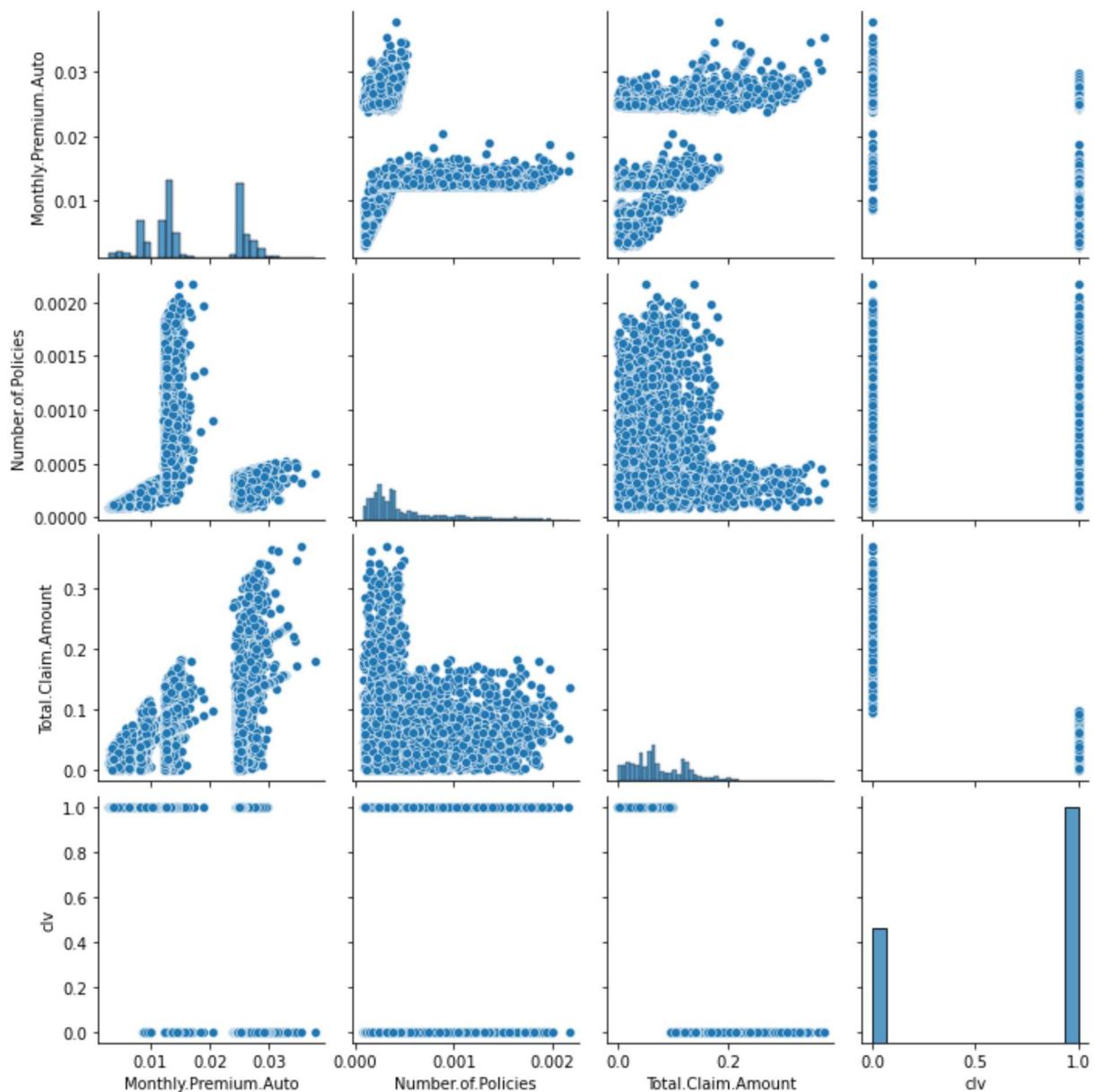
Méthode 3 : régression logistique

On a identifié au début de notre étude qu'un paramètre qui semble avoir aussi un impact sur le CLV est le monthly premium auto. Après avoir effectué un mapping, on voit alors qu'il peut sembler intéressant de réaliser une régression logistique pour prévoir la classe...

In [30]:

```
sb.pairplot(data=df2, vars=['Monthly.Premium.Auto', 'Number.of.Policies', 'Total.Claim.
```

Out[30]:



In [7]:

```
df4 = df2[['Monthly.Premium.Auto', 'clv']]
```

In [8]:

```
df4
```

Out[8]:

	Monthly.Premium.Auto	clv
0	0.024722	0.0
1	0.013293	0.0
2	0.008372	1.0
3	0.013829	1.0
4	0.025905	1.0
...
8625	0.025137	0.0
8626	0.025315	0.0
8627	0.010363	1.0
8628	0.012704	1.0
8629	0.029178	0.0

8630 rows × 2 columns

In [9]:

```
X_train,X_test,y_train,y_test=train_test_split(df4.drop('clv',axis=1),df4['clv'])
```

In [10]:

```
LogReg=LogisticRegression(solver='liblinear')
LogReg.fit(X_train,y_train)
```

Out[10]:

```
LogisticRegression(solver='liblinear')
```

In [15]:

```
y_pred=LogReg.predict(X_test)
LogReg.score(X_test,y_test)
```

Out[15]:

```
0.6714550509731233
```

CONCLUSION PARTIELLE 3 :

Dans le cadre de cette modélisation sous forme de régression logistique, nous constatons que le score obtenu n'est pas très élevé, bien que loin d'être catastrophique. Même s'il semble difficile de proposer ce modèle au client, il nous a tout de même apporté des enseignements importants :

Nous pensions initialement pouvoir modéliser cette dépendance avec une régression logistique, en supposant que lorsque les clients dépassaient une certaine valeur de Monthly Premium Auto, ils passaient d'une classe à une autre. Cependant, cette régression nous a permis de déduire que ceci n'est que partiellement vrai, avec des exceptions non négligeables. C'est pourquoi nous nous orienterons davantage vers une méthode impliquant un arbre de probabilité, car elle prend en compte ce seuil de Monthly Premium Auto tout en réalisant d'autres tests à des niveaux de profondeur variables.

CONCLUSION GENERALE

Rappelons que notre client souhaitait comprendre quels paramètres influencent le Customer Lifetime Value (CLV) et de quelle manière. Notre étude a débuté par une analyse approfondie, étant donné le volume considérable de données fournies par le client. Nous avons commencé par acquérir une compréhension approfondie du CLV (Valeur Client à Vie) pour avoir une vision claire de notre domaine d'étude.

Par la suite, nous avons entrepris de déterminer quels paramètres avaient une influence significative sur notre variable cible, en utilisant notamment la méthode de la matrice de corrélation. Une fois que nous avons identifié les colonnes clés sur lesquelles nous devions nous concentrer, nous avons constaté la nécessité d'approfondir le traitement des données en raison de la variation importante des valeurs dans certaines colonnes par rapport à d'autres. Pour résoudre cette problématique, nous avons mis en place une fonction de normalisation des données.

Ensuite, nous avons procédé à la mise en œuvre de différentes méthodes d'apprentissage automatique (Machine Learning). Étant donné la nature du problème, nous avons opté principalement pour des méthodes d'apprentissage supervisé. En ce qui concerne notre problématique, les principaux paramètres qui ont montré une influence significative sur le CLV sont les suivants :

Monthly Premium Auto : La prime d'assurance mensuelle. Number of Policies : Le nombre de polices d'assurance détenues par le client. Total Claim Amount : Le montant total des sinistres déclarés par le client. Nous avons observé que plus ce montant est bas, plus le CLV tend à être élevé.

Pour répondre à la problématique du client, nous proposons les modèles de prédiction suivants :

Régression Linéaire : Ce modèle permettra au client de prédire la valeur du CLV en fonction du montant total des sinistres déclarés par un client. Arbre de Décision : Ce modèle permettra au client de prédire si la valeur du CLV est considérée comme élevée (classe 1) ou basse (classe 0).

In []: