In [32]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

In [33]:

```python
housing = pd.read_csv('housing.csv')
# Affichage de la taille du dataset (n_lignes and n_colonnes)
print("housing's shape : ", housing.shape)
# Affichage des 10 premières lignes
housing.head()
```

```
housing's shape :  (20640, 10)
```

Out[33]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

In [34]:

```python
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [35]:

```python
missing = housing.isna().sum()
print(missing)
```

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

In [36]:

```python
num_features = housing.select_dtypes(include=[np.number]).columns
print(num_features)

# Print categorical features
cat_features = housing.select_dtypes(include=[np.object]).columns
print(cat_features)

#`ocean_proximity` : attribut catégorique (object à encoder ultérieurement) contenant les catégories suivantes :
housing["ocean_proximity"].value_counts()
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
Index(['ocean_proximity'], dtype='object')
```

Out[36]:

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [37]:

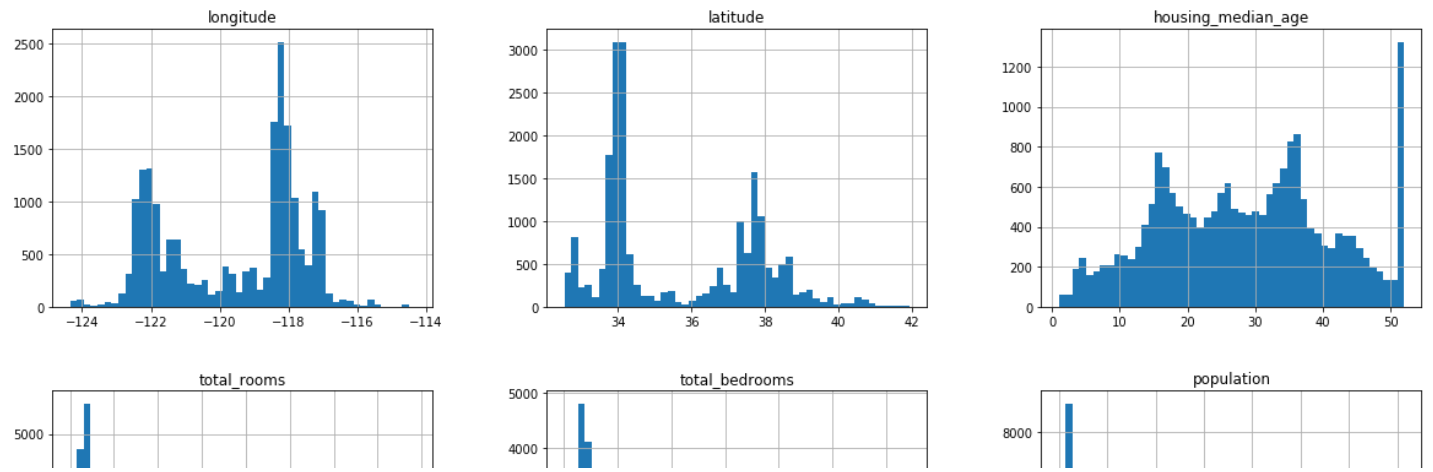
```python
housing.describe()
```

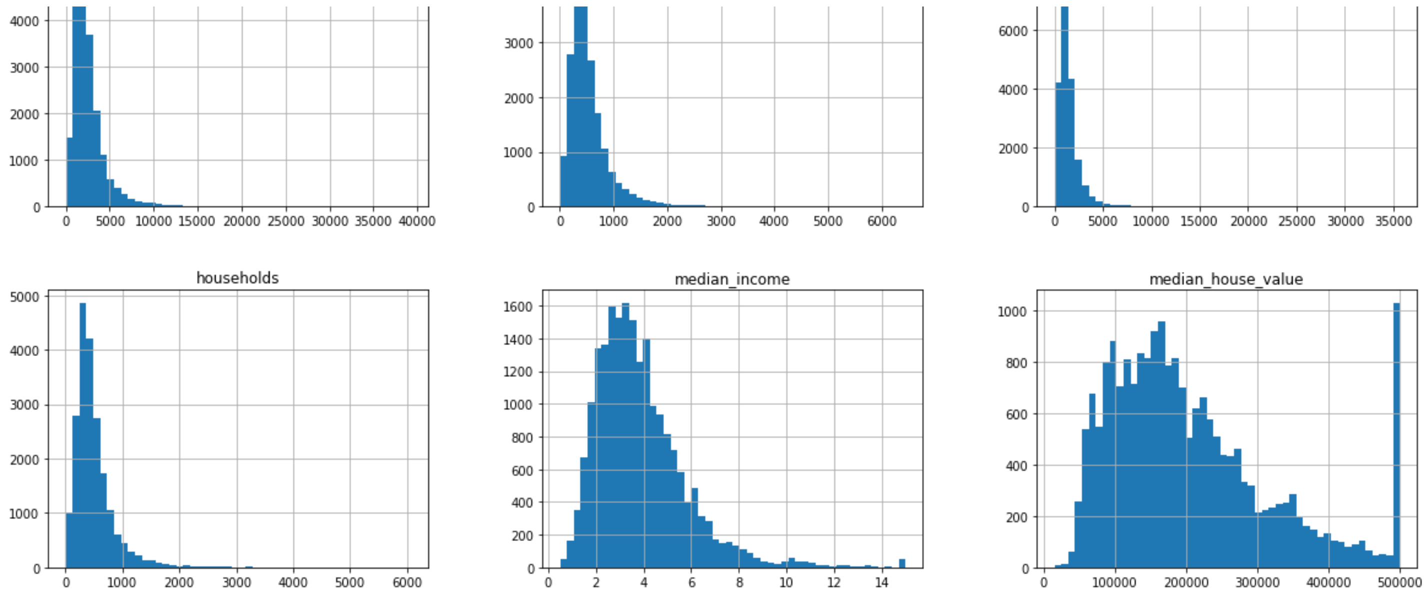
Out[37]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.

In [38]:

```python
# Visualisation des histogrammes des variables numériques
housing.hist(bins=50, figsize=(20,15))
plt.show()
```





In [39]:

```python
# Calculer les coefficients `skewness` des attributs
skewness = housing.skew()
print(skewness)
```

```
longitude             -0.297801
latitude               0.465953
housing_median_age     0.060331
total_rooms            4.147343
total_bedrooms         3.459546
population             4.935858
households             3.410438
median_income          1.646657
median_house_value     0.977763
dtype: float64
```

In [40]:

```python
# Calculer les coefficients `skewness` des attributs
kurtosis = housing.kurtosis()
print(kurtosis)
```

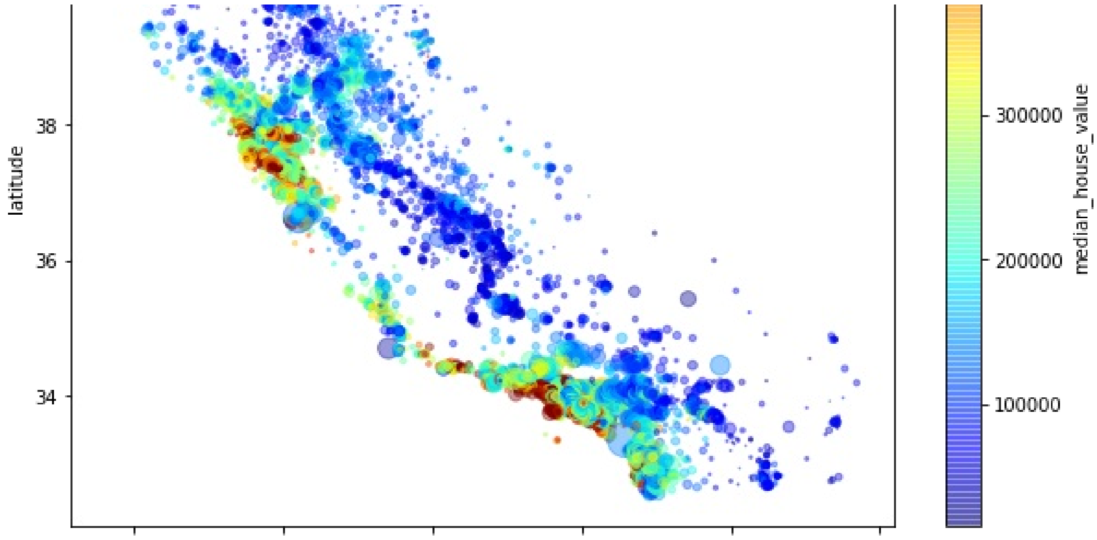
```
longitude             -1.330152
latitude              -1.117760
housing_median_age    -0.800629
total_rooms           32.630927
total_bedrooms        21.985575
population            73.553116
households            22.057988
median_income          4.952524
median_house_value     0.327870
dtype: float64
```

In [41]:

```python
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4, s=housing['populati
on']/100.,
             label='population', figsize=(10, 7), c='median_house_value', cmap=plt.get_c
map(name='jet'), colorbar=True)
plt.legend()
```

Out[41]:

```
<matplotlib.legend.Legend at 0x1c2e54a1358>
```



In [42]:

```
housing[['population', 'median_house_value']].corr()
```

Out[42]:

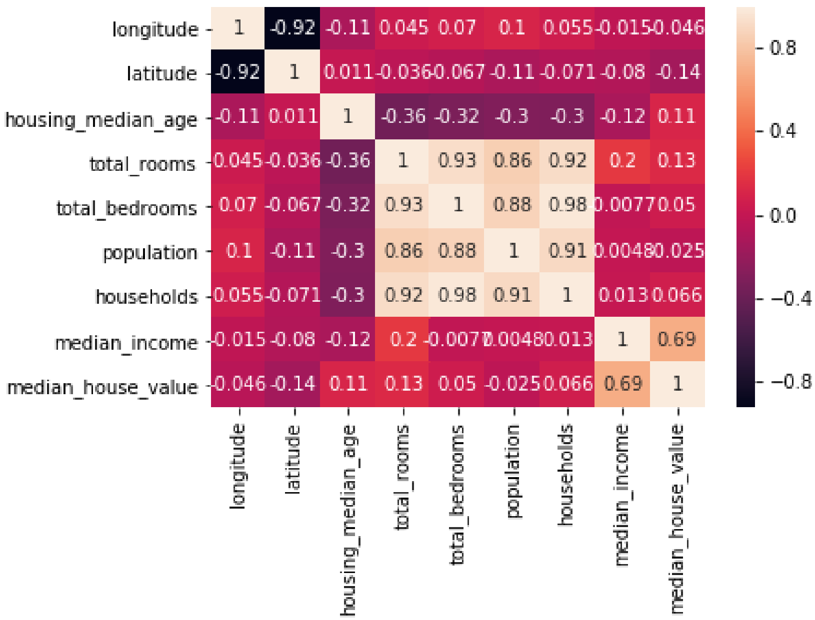
	population	median_house_value
population	1.00000	-0.02465
median_house_value	-0.02465	1.00000

In [43]:

```
corr_matrix = housing.corr()
sns.heatmap(corr_matrix, annot=True)
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2e93606d8>
```



In [44]:

```
# Corrélation de `'median_house_value` avec les autres variables
corr_matrix['median_house_value'].sort_values(ascending=False)
```

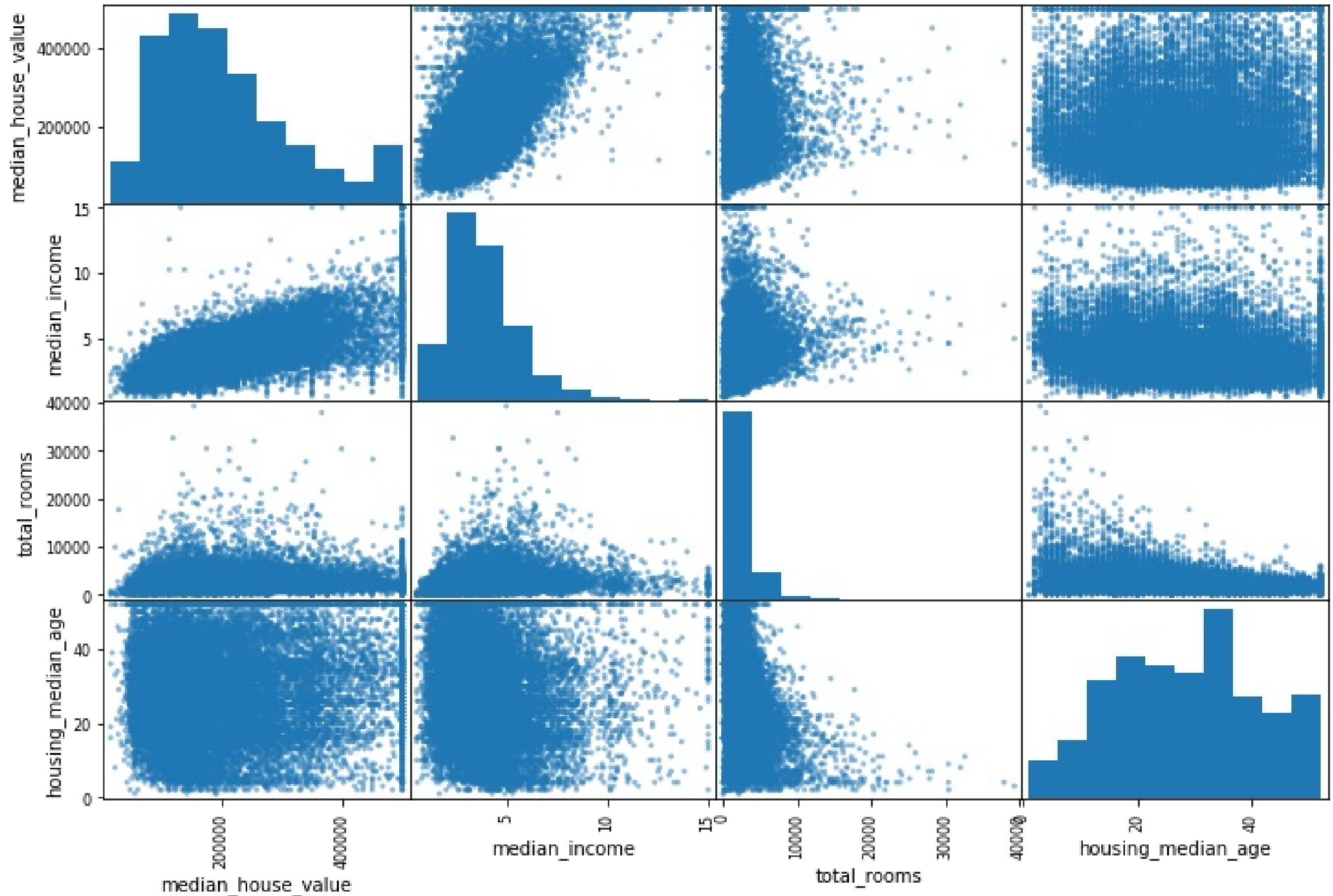
Out[44]:

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
```

```
households             0.065843
total_bedrooms         0.049686
population            -0.024650
longitude             -0.045967
latitude              -0.144160
Name: median_house_value, dtype: float64
```

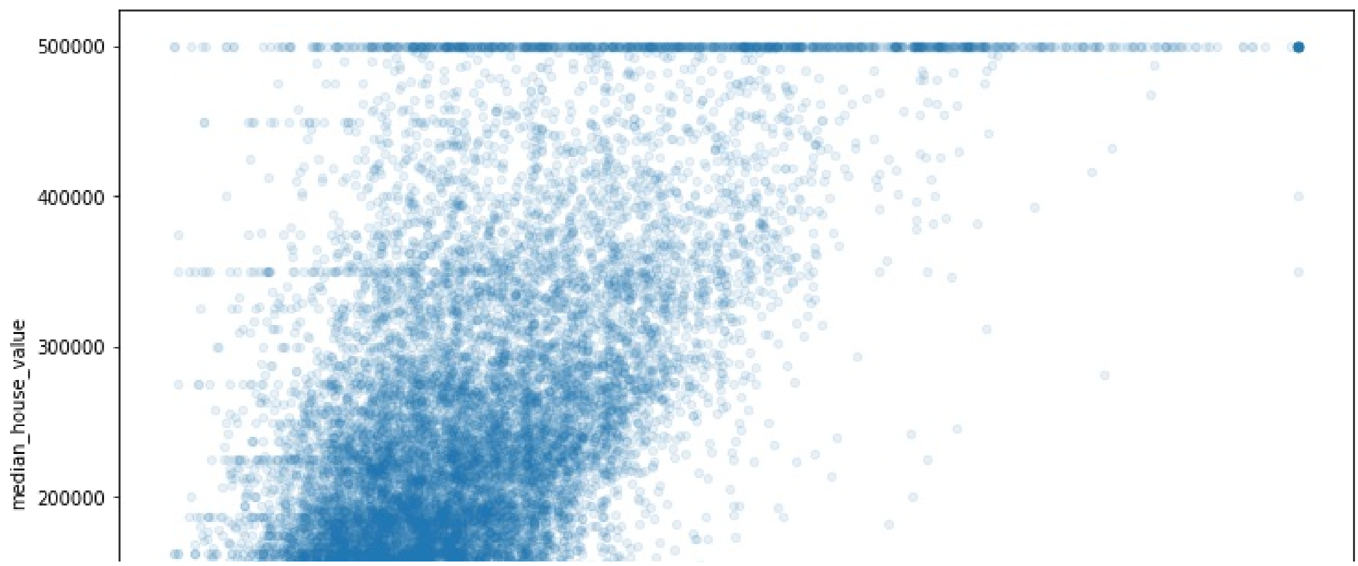
In [45]:

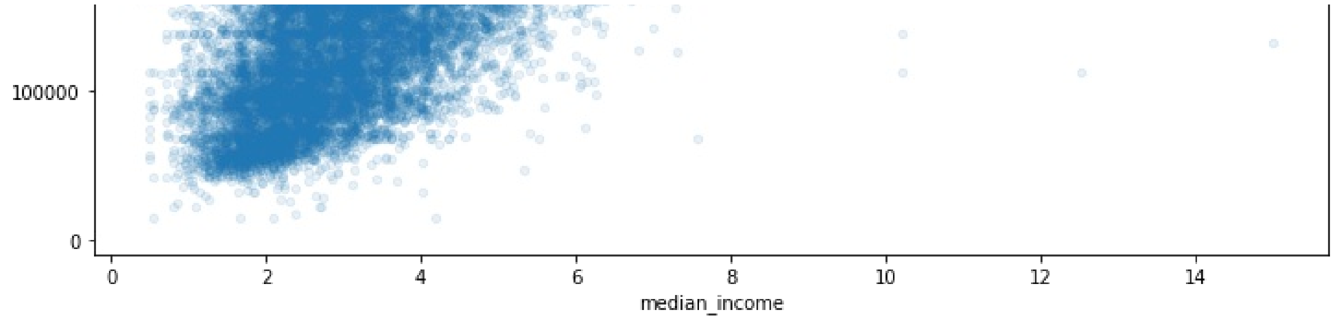
```python
# Prenons les 3 premiers attibuts les plus corrélés avec 'median_house_value'
from pandas.plotting import scatter_matrix
attributes = ['median_house_value', 'median_income', 'total_rooms', 'housing_median_age']
scatter_matrix(frame=housing[attributes], figsize=(12, 8))
plt.show()
```



In [46]:

```python
# ==> L'attribut le plus important dans la prédiction de `median_house_value` est `median_income`
housing.plot(kind='scatter', x='median_income', y='median_house_value', figsize=(12,8), alpha=0.1)
plt.show()
```





In [47]:

```
#Supprimer les lignes dupliquées
housing = housing.drop_duplicates()

# Feature Engineering : Combinaison d'attributs
housing['rooms_per_household'] = housing['total_rooms']/housing['households']
housing['bedrooms_per_room'] = housing['total_bedrooms']/housing['total_rooms']
housing['population_per_household'] = housing['population']/housing['households']

housing.drop(['total_bedrooms'], axis=1, inplace=True)

# vérifiant avec la corrélation
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

Out[47]:

```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

In [48]:

```
# Pour la séparation, on utilise la fonction train_test_split() de Scikit-Learn :
from sklearn.model_selection import train_test_split

X = housing.drop("median_house_value", axis=1) # input variables (X est une dataframe)
y = housing["median_house_value"].to_numpy() # output variable (y est un vecteur)

# `stratify`  permet de s'assurer que les variables y sont équitablement réparties entre
les deux ensembles train et test.
bins = np.linspace(y.min(), y.max(), 100)
y_binned = np.digitize(y, bins)

X_train,X_test,y_train,y_test=train_test_split(X, y,test_size=0.2, shuffle=True, stratif
y=y_binned,random_state=22)

print('X_train:', np.shape(X_train), 'X_test:', np.shape(X_test))
```

```
X_train: (16512, 11) X_test: (4128, 11)
```

In [49]:

```
import pandas as pd
print(pd.__version__)
```

```
1.1.4
```

In [50]:

```
from sklearn.impute import SimpleImputer
```

```
inputer = SimpleImputer(strategy="mean")
inputer.fit_transform([[7, 2, 3], [4, np.nan, 6], [10, 5, 9]])
```

Out[50]:

```
array([[ 7. ,  2. ,  3. ],
       [ 4. ,  3.5,  6. ],
       [10. ,  5. ,  9. ]])
```

In [51]:

```
# Feature scaling : StandardScaler : moyenne = 0 et écart type = 1
from sklearn.preprocessing import StandardScaler
# Génerer une matrice (3x3) avec des valeurs entre 1 et 10
data = np.random.randint(1, 10, (3, 3))
print(data)

scaler = StandardScaler().fit(data)
print("\nmean", scaler.mean_)

data_scaled = scaler.transform(data)
print(data_scaled)

# Nouvelle moyenne 0. et Nouveau écart type 1.
print("\nMoy:", data_scaled.mean(), "Std:", data_scaled.std())
```

```
[[9 6 6]
 [4 1 1]
 [7 2 9]]

mean [6.66666667 3.         5.33333333]
[[ 1.13554995  1.38873015  0.20203051]
 [-1.29777137 -0.9258201  -1.31319831]
 [ 0.16222142 -0.46291005  1.1111678 ]]

Moy: -2.4671622769447922e-17 Std: 0.9999999999999999
```

```
C:\Users\Samar_khlifi\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int32 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\Samar_khlifi\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int32 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [52]:

```
# Encodage des variables catégoriques
from sklearn.preprocessing import OneHotEncoder

data = [["ROUGE"], ["ROUGE"], ["JAUNE"], ["VERT"], ["JAUNE"]]
encoder = OneHotEncoder().fit(data)
data_hot = encoder.transform(data).toarray()

print(data_hot)
print(encoder.categories_)
del data #suppression de data
```

```
[[0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
[array(['JAUNE', 'ROUGE', 'VERT'], dtype=object)]
```

In [53]:

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import PowerTransformer

# Print numerical features
num_features = X_train.select_dtypes(include=[np.number]).columns
```

```python
print(num_features)

# Print categorical features
cat_features = X_train.select_dtypes(include=[np.object]).columns
print(cat_features)

num_pipeline = Pipeline([("imputer", SimpleImputer(strategy="median")),
                         ("transformer", PowerTransformer(method='yeo-johnson', standard
ize=True))])

# le full_pipeline applique num_pipeline aux variables numériques et encode les variables
catégoriques
full_pipeline = ColumnTransformer([("num", num_pipeline, num_features), ("cat", OneHotEn
coder(), cat_features)])

# Apprendre full_pipeline sur training data
full_pipeline = full_pipeline.fit(X_train)

import joblib
joblib.dump(full_pipeline, "DataPreparationModel.pkl")

# Appliquer sur les training data et test data
X_train = full_pipeline.transform(X_train)
X_test = full_pipeline.transform(X_test)

print("\n X_train:", X_train.shape, "X_test:", X_test.shape)

print(X_train[0,:])
features = num_features.to_numpy()
features = np.concatenate((features, ['ocean_1', 'ocean_2', 'ocean_3', 'ocean_4', 'ocean
_5', 'median_house_value']), axis=0)
print(features)
# saugarder dans un fichier CSV les données préparées représentées dataframe
print(np.shape(X_train))
print(np.shape(y_train))
df_train = pd.DataFrame(np.concatenate((X_train, y_train[:, np.newaxis]), axis=1), colum
ns=features)
df_train.to_csv('housing_train.csv', index=False)
df_train.head()

# sauvgarder dans un fichier CSV les données préparées représentées dataframe
print(np.shape(X_test))
print(np.shape(y_test))
df_test = pd.DataFrame(np.concatenate((X_test, y_test[:, np.newaxis]), axis=1), columns=
features)
df_test.to_csv('housing_test.csv', index=False)
df_test.head()
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'population', 'households', 'median_income', 'rooms_per_household',
       'bedrooms_per_room', 'population_per_household'],
      dtype='object')
Index(['ocean_proximity'], dtype='object')

 X_train: (16512, 15) X_test: (4128, 15)
[-1.30857243  1.05876908  0.82883368 -1.18819343 -1.40850033 -0.8230231
 -0.24550082 -1.26183905  1.32210962 -2.15552052  0.          0.
  0.          1.          0.        ]
['longitude' 'latitude' 'housing_median_age' 'total_rooms' 'population'
 'households' 'median_income' 'rooms_per_household' 'bedrooms_per_room'
 'population_per_household' 'ocean_1' 'ocean_2' 'ocean_3' 'ocean_4'
 'ocean_5' 'median_house_value']
(16512, 15)
(16512,)
(4128, 15)
(4128,)
```

Out[53]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	rooms_per_household	bec
0	-	0.684426	-0.736547	-0.482004	-0.302572	-0.120655	-1.726629	-1.045449	

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	rooms_per_household	bec
1	-0.123450	-0.488583	-0.084457	0.830651	0.568531	0.908358	0.315593	-0.080205	
2	0.222210	0.035660	-0.084457	1.146837	0.953889	1.206860	-0.211090	-0.006154	
3	0.617251	-0.591933	1.120496	-0.566029	-0.487707	-0.459440	0.586153	-0.324347	
4	0.543181	-0.756144	-0.993750	-3.363673	-3.809589	-3.937811	1.732088	3.803353	

In [ ]:

In [54]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# chargement des données d'entrainement préparées
df_train = pd.read_csv('housing_train.csv')
df_train.head()
```

Out[54]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	rooms_per_household	bec
0	-1.308572	1.058769	0.828834	-1.188193	-1.408500	-0.823023	-0.245501	-1.261839	
1	0.567871	-0.635401	1.048089	0.058527	-0.136146	0.127886	-0.193363	-0.146672	
2	1.382643	-1.582079	-0.736547	1.393413	1.388602	1.565610	-0.393989	-0.216869	
3	-1.185122	0.890334	-0.243567	0.579701	0.367176	0.547581	0.833637	0.152921	
4	-0.172830	0.660399	1.406965	-0.671689	-0.045317	-0.356069	-1.907723	-0.961184	

In [55]:

```
# extraction de X_train (n_samples, n_features) et y_train (target variable)
X_train = df_train.drop("median_house_value", axis=1)
y_train = df_train["median_house_value"].to_numpy()
print('X_train:', X_train.shape, '; y_train:', np.shape(y_train))
```

```
X_train: (16512, 15) ; y_train: (16512,)
```

In [56]:

```
# chargement des données d'entrainement préparées
df_test = pd.read_csv('housing_test.csv')
# extraction de X_test et y_test
X_test= df_test.drop("median_house_value", axis=1)
y_test = df_test["median_house_value"].to_numpy()
print('X_test:', X_test.shape, '; y_test:', np.shape(y_test))
```

```
X_test: (4128, 15) ; y_test: (4128,)
```

In [57]:

```
# Commençons par le modèle de base
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```python
#Création d'une instance (le modèle lin_reg) par le constructeur LinearRegression()
lin_reg = LinearRegression()
# Apprentissage du modèle par la méthode fit() : Il s'agit d'une approche d'apprentissage
# supervisée puisqu'on utilise lin_reg.fit(X_train, y_train)
lin_reg.fit(X_train, y_train)

#Prédiction des les données d'apprentissage X_train par la méthode .predict()
y_pred = lin_reg.predict(X_train)
#Evaluation de la prédiction obtenue avec les deux métriques R2 et RMSE
rmse = np.sqrt(mean_squared_error(y_train, y_pred))
r2 = r2_score(y_train, y_pred)
print("Training: R2=", r2, " et RMSE=", rmse)

#Prédiction sur les données de test X_test par la méthode .predict()
y_pred = lin_reg.predict(X_test)
#Evaluation de la prédiction obtenue avec les deux métriques R2 et RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print("Testing: R2=", r2, " et RMSE=", rmse)
```

```
Training: R2= 0.6204496969305788  et RMSE= 71112.75132813257
Testing: R2= 0.6267132831766089  et RMSE= 70414.65082312857
```

In [58]:

```python
from sklearn.tree import DecisionTreeRegressor
# Création d'une instance dt_reg par le constructeur DecisionTreeRegressor()
dt_reg = DecisionTreeRegressor()
# Apprentissage du modèle dt_reg par fit
dt_reg.fit(X_train, y_train)

#Pédiction sur X_train par la méthode predict()
y_pred = dt_reg.predict(X_train)
#Evaluation en calculant les métriques R2 et RMSE
rmse = np.sqrt(mean_squared_error(y_train, y_pred))
r2 = r2_score(y_train, y_pred)
print("Training: R2=", r2, " et RMSE=", rmse)

#Prédiction sur X_test par la méthode predict()
y_pred = dt_reg.predict(X_test)
#Evaluation en calculant les métriques R2 et RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print("Testing: R2=", r2, " et RMSE=", rmse)
```

```
Training: R2= 1.0  et RMSE= 0.0
Testing: R2= 0.6320140160867412  et RMSE= 69912.9134418653
```

In [59]:

```python
from sklearn.ensemble import RandomForestRegressor
# Création d'une instance par le constructeur RandomForestRegressor()
rf_reg = RandomForestRegressor()
# Apprentissage du modèle rf_reg par la méthode fit()
rf_reg.fit(X_train, y_train)

#Prédiction sur X_train par la méthode predict()
y_pred = rf_reg.predict(X_train)
#Evaluation e calculant R2 et RMSE
rmse = np.sqrt(mean_squared_error(y_train, y_pred))
r2 = r2_score(y_train, y_pred)
print("Training: R2=", r2, " et RMSE=", rmse)

#Prédiction sur X_test par la méthode predict()
y_pred = rf_reg.predict(X_test)
#Evaluation e calculant R2 et RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print("Testing: R2=", r2, " et RMSE=", rmse)
```

```
C:\Users\Samar_khlifi\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureW
```

```
arning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0
.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
Training: R2= 0.9627110747465382  et RMSE= 22289.62405245616
Testing: R2= 0.7930638184880507  et RMSE= 52427.60002200658
```

In [ ]:

```

```

In [60]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# chargement des données d'entrainement préparées
df_train = pd.read_csv('housing_train.csv')
df_train.head()
```

Out[60]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	rooms_per_household	bec
0	-1.308572	1.058769	0.828834	-1.188193	-1.408500	-0.823023	-0.245501	-1.261839	
1	0.567871	-0.635401	1.048089	0.058527	-0.136146	0.127886	-0.193363	-0.146672	
2	1.382643	-1.582079	-0.736547	1.393413	1.388602	1.565610	-0.393989	-0.216869	
3	-1.185122	0.890334	-0.243567	0.579701	0.367176	0.547581	0.833637	0.152921	
4	-0.172830	0.660399	1.406965	-0.671689	-0.045317	-0.356069	-1.907723	-0.961184	

In [61]:

```
# extraction de X_train et y_train
X_train = df_train.drop("median_house_value", axis=1)
y_train = df_train["median_house_value"].to_numpy()
print('X_train:', X_train.shape, '; y_train:', np.shape(y_train))
```

```
X_train: (16512, 15) ; y_train: (16512,)
```

In [74]:

```
# les modèles testés
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import ExtraTreeRegressor
from sklearn.linear_model import RANSACRegressor
from sklearn.linear_model import HuberRegressor
from sklearn.linear_model import TheilSenRegressor
from sklearn.linear_model import LassoLars
from sklearn.linear_model import BayesianRidge



print(LinearRegression().get_params())
```

```
print(DecisionTreeRegressor().get_params())
print(RandomForestRegressor().get_params())
print(SGDRegressor().get_params())
print(KNeighborsRegressor().get_params())
print(GradientBoostingRegressor().get_params())
print(AdaBoostRegressor().get_params())
print(BaggingRegressor().get_params())
print(ExtraTreeRegressor().get_params())
print(RANSACRegressor().get_params())
print(HuberRegressor().get_params())
print(TheilSenRegressor().get_params())
print(LassoLars().get_params())
print(BayesianRidge().get_params())
```

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize': False}
{'criterion': 'mse', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'mi
n_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_sample
s_split': 2, 'min_weight_fraction_leaf': 0.0, 'presort': False, 'random_state': None, 'sp
litter': 'best'}
{'bootstrap': True, 'criterion': 'mse', 'max_depth': None, 'max_features': 'auto', 'max_l
eaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_
leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 'warn'
, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': F
alse}
{'alpha': 0.0001, 'average': False, 'early_stopping': False, 'epsilon': 0.1, 'eta0': 0.01
, 'fit_intercept': True, 'l1_ratio': 0.15, 'learning_rate': 'invscaling', 'loss': 'square
d_loss', 'max_iter': None, 'n_iter': None, 'n_iter_no_change': 5, 'penalty': 'l2', 'power
_t': 0.25, 'random_state': None, 'shuffle': True, 'tol': None, 'validation_fraction': 0.1
, 'verbose': 0, 'warm_start': False}
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_j
obs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
{'alpha': 0.9, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1, 'loss': '
ls', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease
': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_w
eight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_iter_no_change': None, 'presort': 'aut
o', 'random_state': None, 'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'v
erbose': 0, 'warm_start': False}
{'base_estimator': None, 'learning_rate': 1.0, 'loss': 'linear', 'n_estimators': 50, 'ran
dom_state': None}
{'base_estimator': None, 'bootstrap': True, 'bootstrap_features': False, 'max_features':
1.0, 'max_samples': 1.0, 'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_
state': None, 'verbose': 0, 'warm_start': False}
{'criterion': 'mse', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, '
min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samp
les_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': None, 'splitter': 'random
'}
{'base_estimator': None, 'is_data_valid': None, 'is_model_valid': None, 'loss': 'absolute
_loss', 'max_skips': inf, 'max_trials': 100, 'min_samples': None, 'random_state': None, '
residual_threshold': None, 'stop_n_inliers': inf, 'stop_probability': 0.99, 'stop_score':
inf}
{'alpha': 0.0001, 'epsilon': 1.35, 'fit_intercept': True, 'max_iter': 100, 'tol': 1e-05,
'warm_start': False}
{'copy_X': True, 'fit_intercept': True, 'max_iter': 300, 'max_subpopulation': 10000, 'n_j
obs': None, 'n_subsamples': None, 'random_state': None, 'tol': 0.001, 'verbose': False}
{'alpha': 1.0, 'copy_X': True, 'eps': 2.220446049250313e-16, 'fit_intercept': True, 'fit_
path': True, 'max_iter': 500, 'normalize': True, 'positive': False, 'precompute': 'auto',
'verbose': False}
{'alpha_1': 1e-06, 'alpha_2': 1e-06, 'compute_score': False, 'copy_X': True, 'fit_interce
pt': True, 'lambda_1': 1e-06, 'lambda_2': 1e-06, 'n_iter': 300, 'normalize': False, 'tol'
: 0.001, 'verbose': False}
```

In [75]:

```
# K-fold cross-validation et GridSearchCV
pipelines = []
params = []
names = []

#
# ajouter LinearRegression
pipelines.append(Pipeline([('clf', LinearRegression())])) ### LinearRegression
params.append({'clf__normalize':[True]})
```

```
names.append('LinearRegression')

# ajouter DecisionTreeRegressor
pipelines.append(Pipeline([('clf', DecisionTreeRegressor())])) ## DecisionTreeRegressor
params.append({'clf__max_depth':np.linspace(5, 15, 5)})
names.append('DecisionTreeRegressor')

# ajouter RandomForestRegressor
pipelines.append(Pipeline([('clf', RandomForestRegressor())])) ## RandomForestRegressor
params.append({'clf__n_estimators': [50,100,200]})
names.append('RandomForestRegressor')

# ajouter SGDRegressor
pipelines.append(Pipeline([('clf', SGDRegressor())])) ### SGDRegressor
params.append({'clf__average': [True]})
names.append('SGDRegressor')

# ajouter KNeighborsRegressor
pipelines.append(Pipeline([('clf', KNeighborsRegressor())])) ### KNeighborsRegressor
params.append({'clf__n_neighbors':np.array([10])})
names.append('KNeighborsRegressor')

# ajouter GradientBoostingRegressor
pipelines.append(Pipeline([('clf', GradientBoostingRegressor())])) ### GradientBoostingRe
gressor
params.append({'clf__n_estimators':[50,100,200]})
names.append('GradientBoostingRegressor')

# ajouter AdaBoostRegressor
pipelines.append(Pipeline([('clf', AdaBoostRegressor())])) ### AdaBoostRegressor
params.append({'clf__n_estimators':[100]})
names.append('AdaBoostRegressor')

# ajouter BaggingRegressor
pipelines.append(Pipeline([('clf', BaggingRegressor())])) ### BaggingRegressor
params.append({'clf__n_estimators':[50]})
names.append('BaggingRegressor')

# ajouter ExtraTreeRegressor
pipelines.append(Pipeline([('clf', ExtraTreeRegressor())])) ### ExtraTreeRegressor
params.append({'clf__max_depth':np.linspace(5, 15, 5)})
names.append('ExtraTreeRegressor')

# ajouter RANSACRegressor
pipelines.append(Pipeline([('clf', RANSACRegressor())])) ### RANSACRegressor
params.append({'clf__max_trials':[400]})
names.append('RANSACRegressor')

# ajouter HuberRegressor
pipelines.append(Pipeline([('clf', HuberRegressor())])) ### HuberRegressor
params.append({'clf__max_iter':[50]})
names.append('HuberRegressor')


# ajouter TheilSenRegressor
pipelines.append(Pipeline([('clf', TheilSenRegressor())])) ### TheilSenRegressor
params.append({'clf__max_iter':[500]})
names.append('TheilSenRegressor')

# ajouter LassoLars
pipelines.append(Pipeline([('clf', LassoLars())])) ### LassoLars
params.append({'clf__max_iter':[1000]})
names.append('LassoLars')

# ajouter BayesianRidge
pipelines.append(Pipeline([('clf', BayesianRidge())])) ### BayesianRidge
params.append({'clf__n_iter':[600]})
names.append('BayesianRidge')
```

In [76]:

```
# l'enraînement avec cross-validation
```

```python
#n_jobs = -1 signifie que le calcul sera distribué sur tous les CPU de l'ordinateur.

from sklearn.model_selection import KFold, GridSearchCV, cross_val_score

def model(pipeline, parameters, name, X, y):
    cv = KFold(n_splits=5, shuffle=True, random_state=32)
    grid_obj = GridSearchCV(estimator=pipeline, param_grid=parameters, cv=cv, scoring='r
2', n_jobs=-1)
    grid_obj.fit(X,y)
    print(name, 'R2:', grid_obj.best_score_)
    estimator = grid_obj.best_estimator_
    estimator.fit(X,y) # training sur tout training dataset
    return estimator
estimators = []
for i in range(len(pipelines)):
    estimators.append(model(pipelines[i], params[i], names[i], X_train, y_train))
```

```
LinearRegression R2: 0.6189009528164291
DecisionTreeRegressor R2: 0.7148133697624193
RandomForestRegressor R2: 0.8094969486473614
```

```
C:\Users\Samar_khlifi\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradien
t.py:166: FutureWarning: max_iter and tol parameters have been added in SGDRegressor in 0
.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None,
max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default
tol will be 1e-3.
  FutureWarning)
C:\Users\Samar_khlifi\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradien
t.py:166: FutureWarning: max_iter and tol parameters have been added in SGDRegressor in 0
.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None,
max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default
tol will be 1e-3.
  FutureWarning)
```

```
SGDRegressor R2: 0.6154704096748046
KNeighborsRegressor R2: 0.7450883985102568
GradientBoostingRegressor R2: 0.8083426664943411
AdaBoostRegressor R2: 0.46636049770680876
BaggingRegressor R2: 0.8064123188252824
ExtraTreeRegressor R2: 0.6745321018819196
RANSACRegressor R2: 0.43232292900354474
HuberRegressor R2: 0.61045171529856
TheilSenRegressor R2: 0.6046702657093727
LassoLars R2: 0.6178317737784579
BayesianRidge R2: 0.6188661030188772
```

In [77]:

```python
from sklearn.metrics import mean_squared_error, r2_score

# chargement des données d'entrainement préparées
df_test = pd.read_csv('housing_test.csv')

# extraction de X_test et y_test
X_test= df_test.drop("median_house_value", axis=1)
y_test = df_test["median_house_value"].to_numpy()
print('X_test:', X_test.shape, '; y_test:', np.shape(y_test))

# Evaluation

for i, estimator in enumerate(estimators):
    print('\nPerformance :', names[i])
    y_pred = estimator.predict(X_test)
    print('\n mean_squared_error :', mean_squared_error(y_test, y_pred))
    print('\n r2_score :', r2_score(y_test, y_pred))
```

```
X_test: (4128, 15) ; y_test: (4128,)

Performance : LinearRegression

 mean_squared_error : 4958856257.3338175
```

```
 r2_score : 0.6266656113228902

Performance : DecisionTreeRegressor

 mean_squared_error : 3847097459.015217

 r2_score : 0.7103659183670459

Performance : RandomForestRegressor

 mean_squared_error : 2431591680.454908

 r2_score : 0.8169342391821888

Performance : SGDRegressor

 mean_squared_error : 4973082631.939186

 r2_score : 0.6255945589287959

Performance : KNeighborsRegressor

 mean_squared_error : 3204385786.1675487

 r2_score : 0.7587533603549765

Performance : GradientBoostingRegressor

 mean_squared_error : 2547874517.2325354

 r2_score : 0.8081797241228318

Performance : AdaBoostRegressor

 mean_squared_error : 6504225846.211189

 r2_score : 0.510320312970982

Performance : BaggingRegressor

 mean_squared_error : 2482880977.932025

 r2_score : 0.8130728531032962

Performance : ExtraTreeRegressor

 mean_squared_error : 4163550169.9573755

 r2_score : 0.686541335992818

Performance : RANSACRegressor

 mean_squared_error : 6174779839.7036

 r2_score : 0.5351231136691725

Performance : HuberRegressor

 mean_squared_error : 5071709057.123098

 r2_score : 0.6181693313677048

Performance : TheilSenRegressor

 mean_squared_error : 5233811613.771304

 r2_score : 0.6059652149850707

Performance : LassoLars

 mean_squared_error : 4957467719.866949
```

```
 r2_score : 0.6267701492968599

Performance : BayesianRidge

 mean_squared_error : 4959579429.772195

 r2_score : 0.6266111662802878
```

In [78]:

```
# Serialize final models
import joblib
for i, estimator in enumerate(estimators):
    joblib.dump(estimator, names[i]+".pkl")


# chargement du modèle linear regression
# model = joblib.load(names[0]+"pkl")
```

In [ ]: