0 - Modules ¶

Entrée [1]:

```
# ###################################
#
#  much of the cod is stored in
#  my package for more readability
#
# ###################################
#from mypackage import ploter_bis as plt_bis
from mypackage import ploter
from mypackage import data_processor as dp
from mypackage import mydataloader as dl
```

Entrée [2]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import svm
from sklearn import neighbors
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
print(tscv)
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
import datetime
import seaborn as sns
import matplotlib.pyplot as plt
from pickle import dump
import platform
```

```
TimeSeriesSplit(max_train_size=None, n_splits=5)
```

1 - Data Preparation

1.1 - Load Data

Entrée [3]:

```
! mkdir data

# ########## Chose the periode

#years = [2020,2019,2018,2017,2016,2015,2014]
#years = [2020]
years = [2020,2019,2018,2017]


# ########## Chose the area with departement code

#departements = ['75']
#departements = ['75','92','93','94','77','78','91','95']
departements = ['75','92','93','94','95']


data = dl.get_market_data(years = years,departements=departements,top_cities=None)
```

```
https://www.dropbox.com/s/rmuej0caz03hqlk/valeursfoncieres-2020.zip?dl=1 (ht
tps://www.dropbox.com/s/rmuej0caz03hqlk/valeursfoncieres-2020.zip?dl=1)
valeursfoncieres-2020 :
        -initial shape (2459560, 12)
        -final shape (84613, 17)
https://www.dropbox.com/s/i0lgj7gc70h33sr/valeursfoncieres-2019.zip?dl=1 (ht
tps://www.dropbox.com/s/i0lgj7gc70h33sr/valeursfoncieres-2019.zip?dl=1)
valeursfoncieres-2019 :
        -initial shape (1017154, 12)
        -final shape (38769, 17)
https://www.dropbox.com/s/sbcoqaqja94qlm4/valeursfoncieres-2018.zip?dl=1 (ht
tps://www.dropbox.com/s/sbcoqaqja94qlm4/valeursfoncieres-2018.zip?dl=1)
valeursfoncieres-2018 :
        -initial shape (2339002, 12)
        -final shape (81709, 17)
https://www.dropbox.com/s/wydzvm25xu4ubtl/valeursfoncieres-2017.zip?dl=1 (ht
tps://www.dropbox.com/s/wydzvm25xu4ubtl/valeursfoncieres-2017.zip?dl=1)
valeursfoncieres-2017 :
        -initial shape (3361073, 12)
        -final shape (103387, 17)

=> Final Data Frame shape: (308478, 17)
```

1.2 - Features engineering with special encoding

Entrée [4]:

```
X_train,X_test,y_train,y_test = dp.feature_engineering(data)
```

Entrée [5]:

```
1 X_train.head()
```

Out[5]:

	Date mutation	Type de voie	Voie	Code postal	Commune	Code departement	Type local	dep	
0	0.000000	RUE	EMILE GOEURY	0.872599	GAGNY	0.90	Maison	0.90	
1	0.001103	BD	GABRIEL PERI	0.825662	MALAKOFF	0.85	Appartement	0.85	MAI
2	0.001103	RUE	MAUREPAS	0.925284	THIAIS	0.95	Appartement	0.95	THIAISR
3	0.001103	RUE	DE PICPUS	0.000527	PARIS 12	0.00	Appartement	0.00	
4	0.001103	RUE	GAUTHEY	0.000766	PARIS 17	0.00	Appartement	0.00	17RUE2

5 rows × 28 columns

2 - Experimentation

2.1 - Helper function for Model Selection

> Based on Time Series Cross Validation

Entrée [6]:

```
1  def model_selection(models,X,y,verbose=True):
2      i=1
3      model_score = []
4      for name,model in models.items():
5          scores = cross_val_score(model, X,y, cv=tscv)
6          if verbose:
7              print(f"{name} | score : {scores.sum()/5}")
8          model_score.append((scores.sum()/5,model))
9          i+=1
10     best_model = sorted(model_score, reverse=True)[0][1]
11     print('\n######## Best Model ########\n\t%s'%str(best_model))
12     return best_model
```

2.2 - Model preparation

Entrée [7]:

```
ols = linear_model.LinearRegression()
ridge = linear_model.Ridge(alpha=.1)
lasso = linear_model.Lasso(alpha=0.1)
bayesian_ridge = linear_model.BayesianRidge()
svr = svm.SVR() # >== scale very badly
rf = RandomForestRegressor(max_depth=10,min_samples_leaf=10, random_state=0)
gbreg = GradientBoostingRegressor()
```

2.3 - Cross validation with Initial Numerical Features

Entrée [8]:

```
models = {'OLS regression' : ols,
          'Ridge regression' : ridge,
          'Lasso regression' : lasso,
          'Bayesian Ridge regression' : bayesian_ridge
          }


features_basic = ['surface','pieces','terrain']

model = model_selection(models,X_train[features_basic],y_train)

model.fit(X_train[features_basic],y_train)
```

```
OLS regression | score : 0.06867106768673595
Ridge regression | score : 0.07021216911326715
Lasso regression | score : -0.0002792843537354095
Bayesian Ridge regression | score : 0.06874117572809546

######## Best Model ########
        Ridge(alpha=0.1)
```

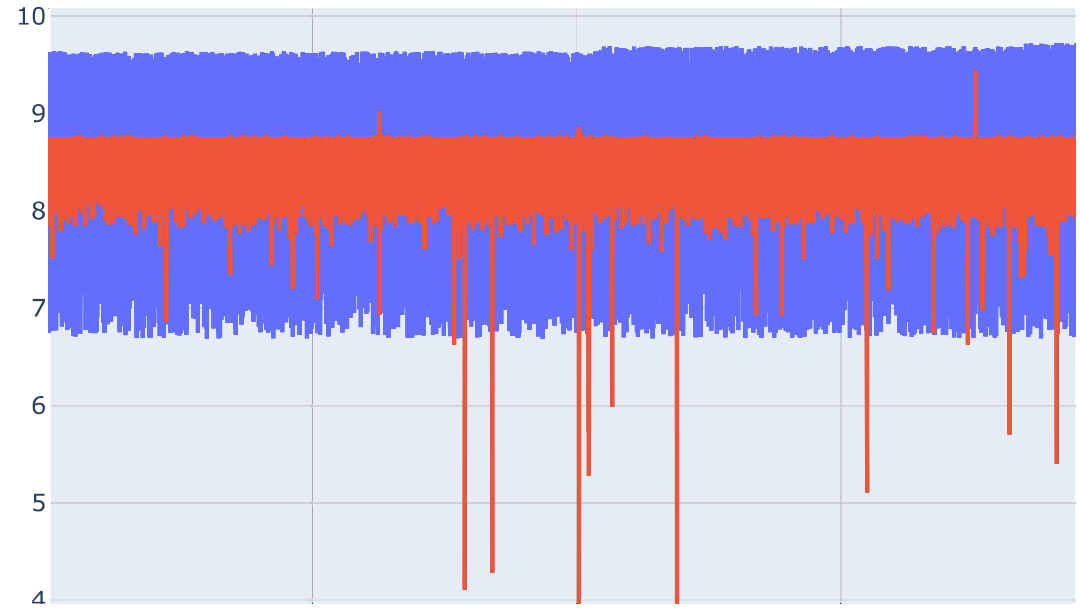
Out[8]:

```
Ridge(alpha=0.1)
```

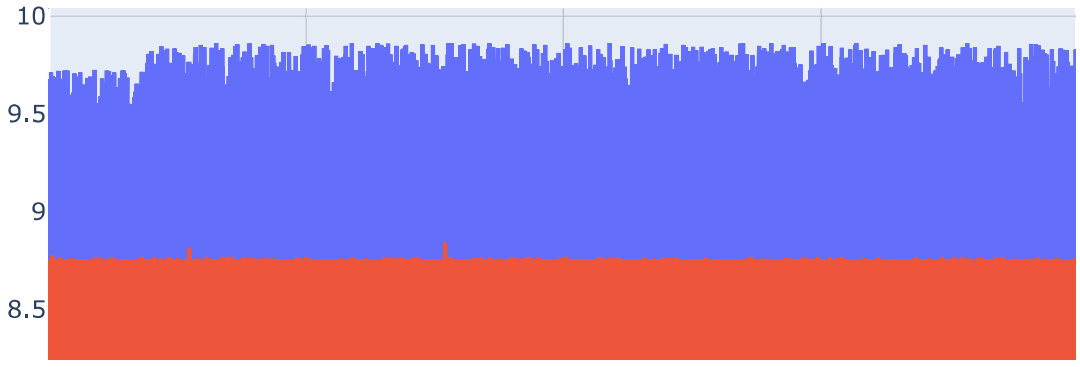
Entrée [9]:

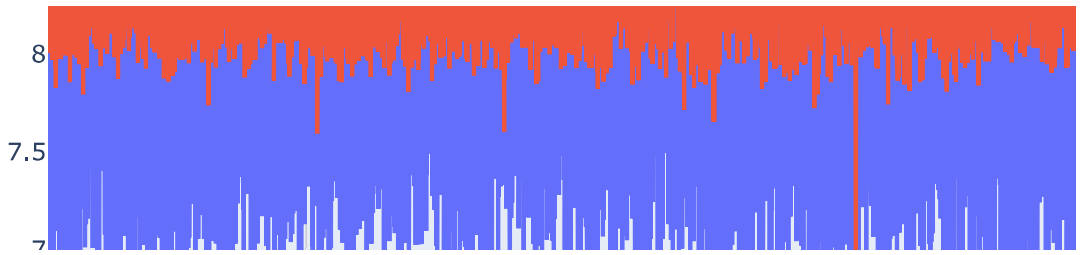
```
1 ploter.check_model_performances(X_train[features_basic],y_train,model,show=True)
2 ploter.check_model_performances(X_test[features_basic],y_test,model,True)
```

```
MSE : 0.28984074347423056
R2 : 0.08038861510301676
```




```
MSE : 0.3377734241319092
R2 : 0.04668425165711776
```





2.4 - Cross validation with Encoded Categorical Features

Entrée [10]:

```
models = {'OLS regression' : ols,
          'Ridge regression' : ridge,
          'Lasso regression' : lasso,
          'Bayesian Ridge regression' : bayesian_ridge,
          'RandomForestRegressor' : rf,
          'Gradient Boosting Regressor':gbreg
          }
```

2.4.1 - Cross validation with 'encodage_voie' as a single feature

Entrée [11]:

```
features_augmented = ['encodage_voie']


model = model_selection(models,X_train[features_augmented],y_train)
model.fit(X_train[features_augmented],y_train)
```

```
OLS regression | score : 0.7555434557992283
Ridge regression | score : 0.7555425728932585
Lasso regression | score : -0.0002792843537354095
Bayesian Ridge regression | score : 0.7555433809033857
RandomForestRegressor | score : 0.7501129674720686
Gradient Boosting Regressor | score : 0.7542752865616117

######## Best Model ########
        LinearRegression()
```

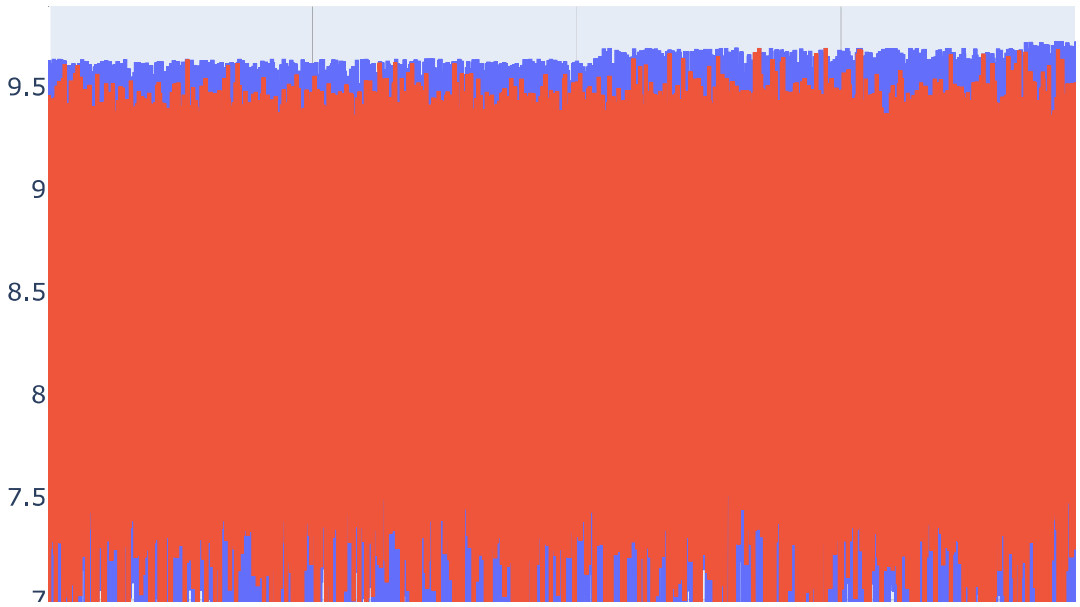
Out[11]:

```
LinearRegression()
```

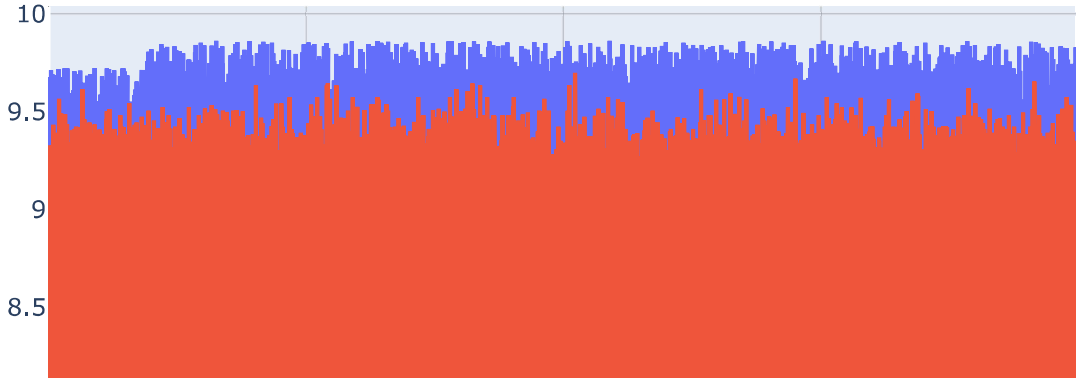
Entrée [12]:

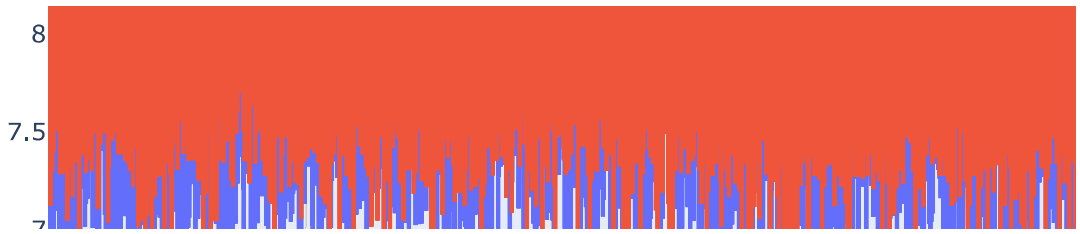
```
1 ploter.check_model_performances(X_train[features_augmented],y_train,model,show=True)
2 ploter.check_model_performances(X_test[features_augmented],y_test,model,True)
```

```
MSE : 0.07675480257269243
R2 : 0.7564711246414402
```




```
MSE : 0.12393726168387909
R2 : 0.6502053301754325
```





2.4.2 - Cross validation with all encoded features

Entrée [13]:

```
features_augmented = ['surface','pieces','encodage_voie','encodage_piece',
                      'encodage_type_voie','encodage_ville','encodage_departement',
                       'terrain','id_local']

model = model_selection(models,X_train[features_augmented],y_train)
model.fit(X_train[features_augmented],y_train)
```

```
OLS regression | score : 0.7640868741384634
Ridge regression | score : 0.7645821376569056
Lasso regression | score : 0.2421542690618111
Bayesian Ridge regression | score : 0.7641444727587106
RandomForestRegressor | score : 0.7817626773523969
Gradient Boosting Regressor | score : 0.7817024804568219

######## Best Model ########
        RandomForestRegressor(max_depth=10, min_samples_leaf=10, random_stat
e=0)
```

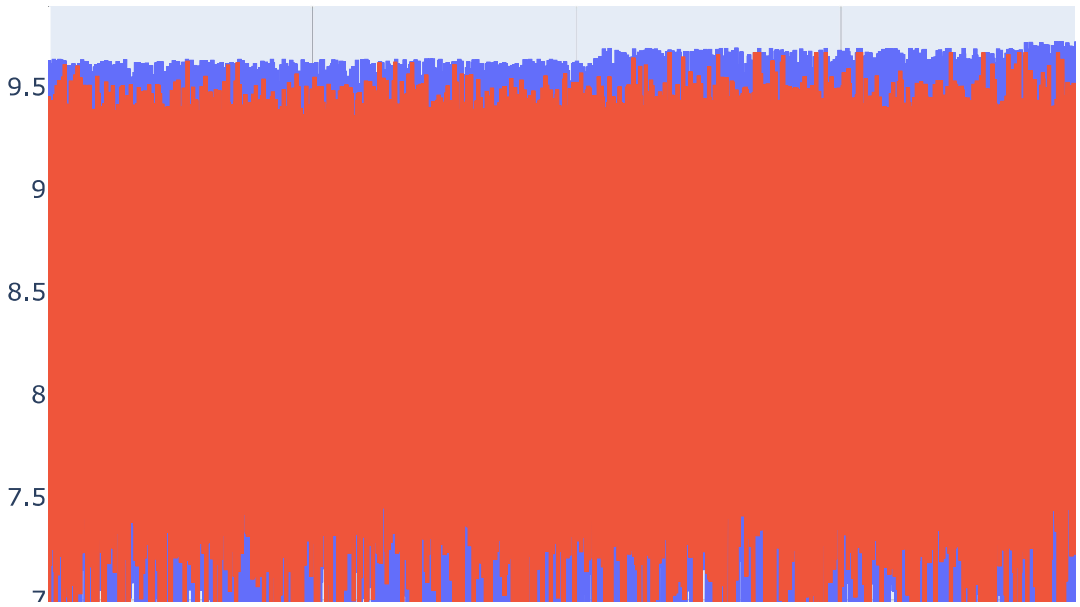
Out[13]:

```
RandomForestRegressor(max_depth=10, min_samples_leaf=10, random_state=0)
```

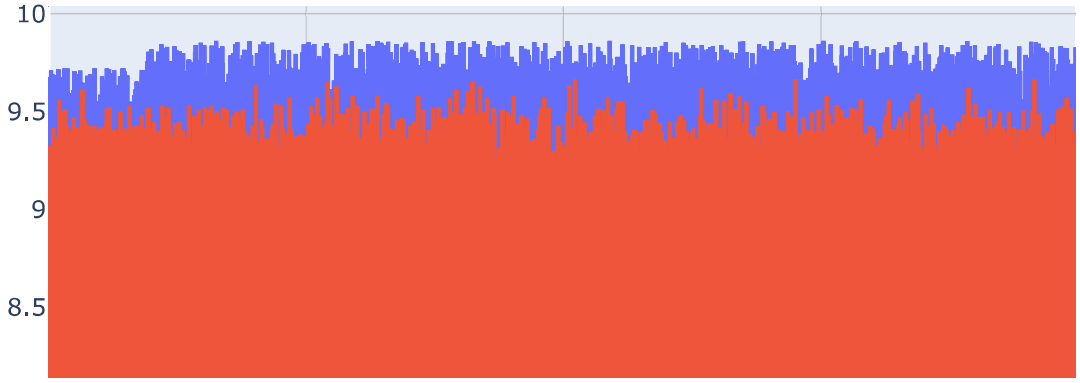
Entrée [14]:

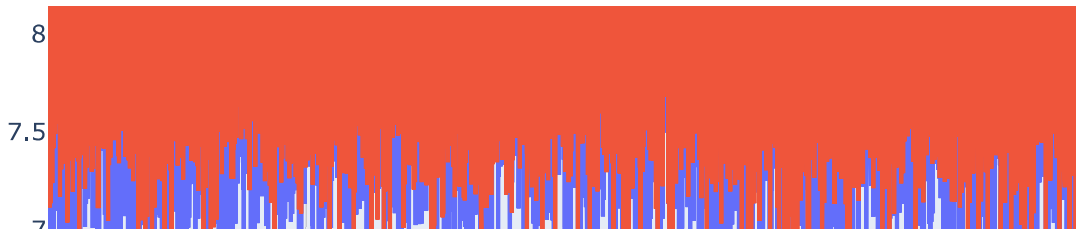
```
1 ploter.check_model_performances(X_train[features_augmented],y_train,model,show=True)
2 ploter.check_model_performances(X_test[features_augmented],y_test,model,True)
```

```
MSE : 0.0641745278549031
R2 : 0.7963860231368594
```




```
MSE : 0.10825692200192139
R2 : 0.6944607798058876
```





2.4.3 - Cross validation with 5 encoded features

These features will be used for ease of future model deployment.

- surface
- pieces
- encodage_voie
- terrain
- id_local

Entrée [15]:

```
features_augmented = ['surface','pieces','encodage_voie',
                      'terrain','id_local']

model = model_selection(models,X_train[features_augmented],y_train)
model.fit(X_train[features_augmented],y_train)
```

```
OLS regression | score : 0.7585855800205203
Ridge regression | score : 0.7590834569460954
Lasso regression | score : -0.0002792843537354095
Bayesian Ridge regression | score : 0.7586198729946213
RandomForestRegressor | score : 0.7785047890841998
Gradient Boosting Regressor | score : 0.7784809708798172

######## Best Model ########
        RandomForestRegressor(max_depth=10, min_samples_leaf=10, random_stat
e=0)
```

Out[15]:

```
RandomForestRegressor(max_depth=10, min_samples_leaf=10, random_state=0)
```

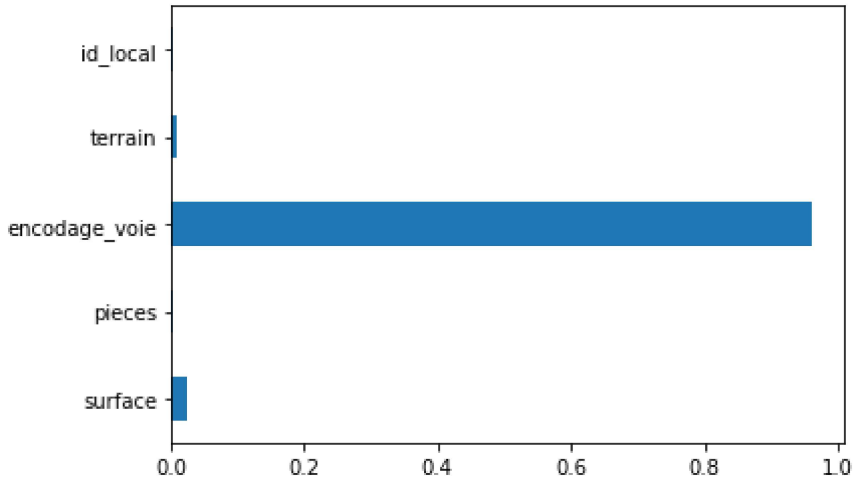
3 - Feature importance

Entrée [16]:

```
try :
    feat_importances = pd.Series(model.feature_importances_, index=features_augmented)
except :
    feat_importances = pd.Series(model.coef_, index=features_augmented)
feat_importances.plot(kind='barh')
```

Out[16]:

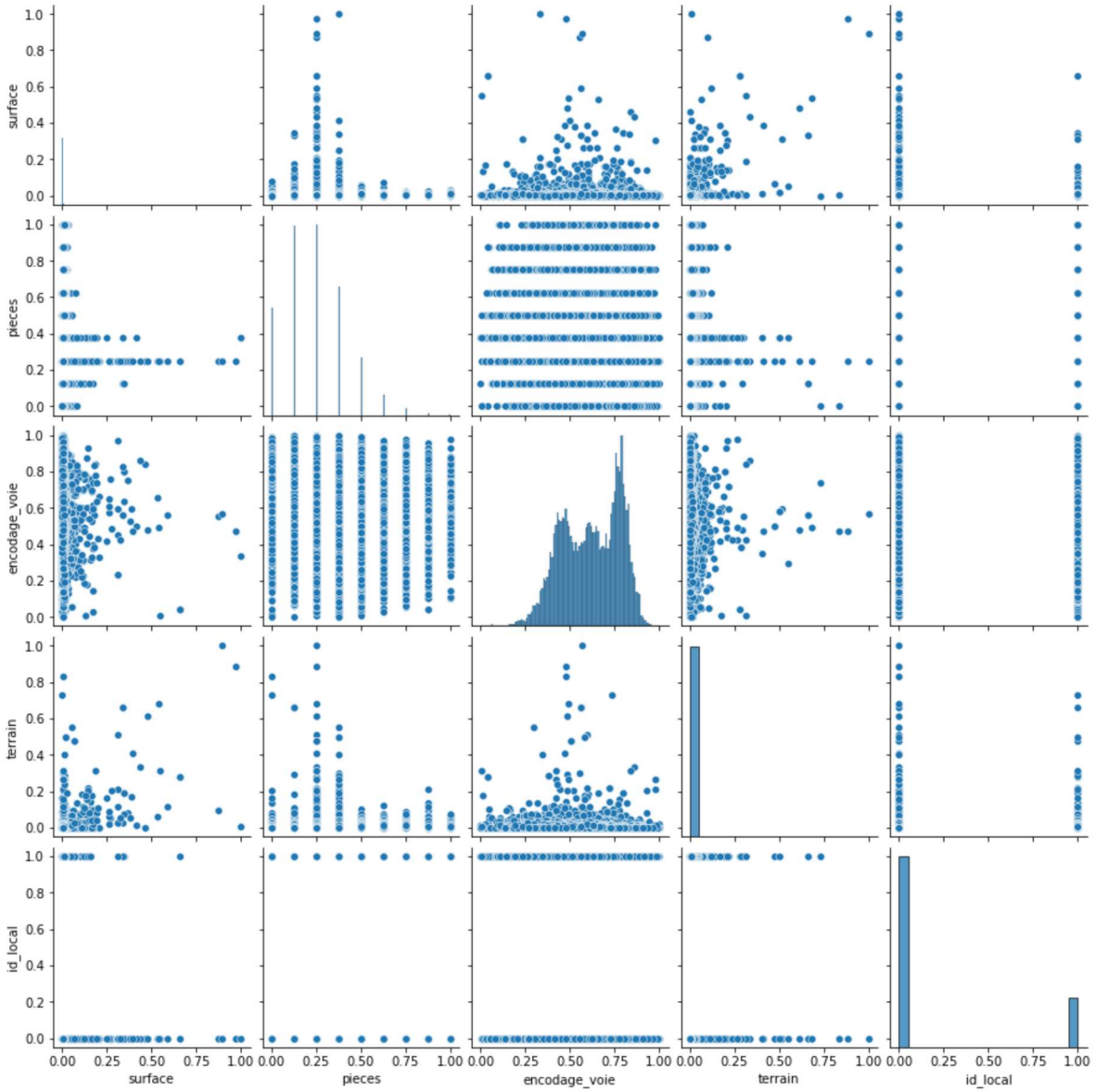
<AxesSubplot:>



Entrée [17]:

```
D = X_train[features_augmented]
D['y'] = y_train

sns.pairplot(D,vars=D.columns[:-1])
plt.show()
```



4 - Save the model

Entrée [18]:

```
# #### not working with docker and my configuration

#dump(model, open('model.pkl', 'wb'))
```

4 - Investigate errors on Train Set

Entrée [19]:

```
train_info = X_train[['Commune','surface','pieces']]
predictions_train = model.predict(X_train[features_augmented])

final_predictions_train = pd.concat([train_info.reset_index(),y_train.reset_index().dro
.drop(columns='index')


final_predictions_train.columns = ['commune','surface','pieces','target','pred']

final_predictions_train[['target','pred']] = np.exp(final_predictions_train[['target',

final_predictions_train['error'] = abs(final_predictions_train.target-final_predictions

final_predictions_train.sort_values(by=['error'],ascending=False).tail(10000)

```

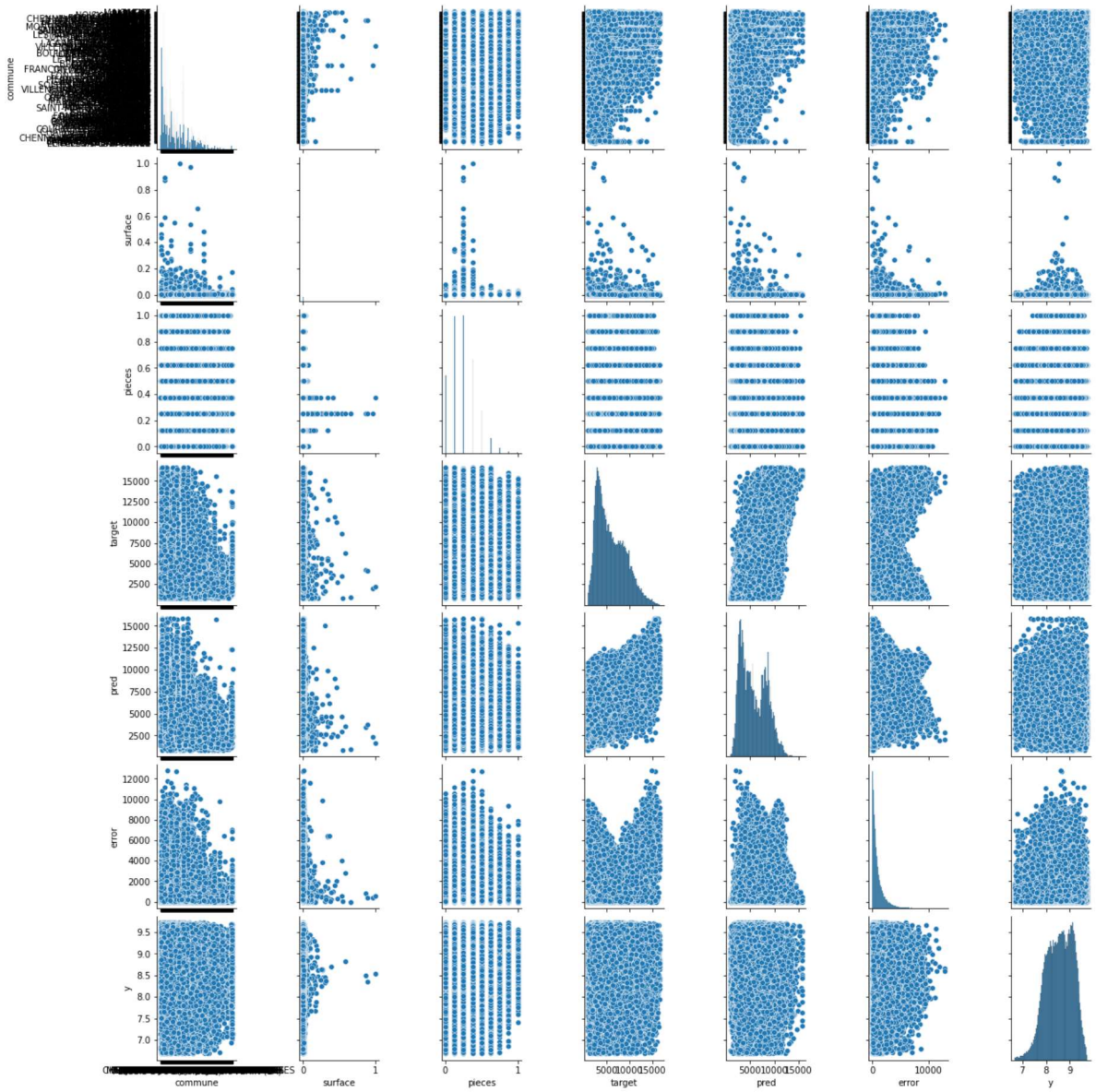
	commune	surface	pieces	target	pred	error
148835	AUBERVILLIERS	0.000531	0.125	4318.181818	4359.181756	40.999938
64262	LOUVRES	0.002293	0.375	2470.588235	2429.602135	40.986101
77895	LE PLESSIS-ROBINSON	0.000979	0.125	6175.263158	6134.281745	40.981413
83540	ARGENTEUIL	0.001063	0.125	2585.365854	2626.339262	40.973408
19448	NEUILLY-PLAISANCE	0.001203	0.125	3956.521739	3997.486131	40.964392
...	...	...	...	...	...	...
116895	NOISY-LE-GRAND	0.001734	0.250	3338.461538	3338.479756	0.018217
138005	ST GRATIEN	0.002405	0.375	2674.157303	2674.145141	0.012162
201684	EPINAY SUR SEINE	0.002349	0.500	2402.298851	2402.292949	0.005902
70505	CORMEILLES-EN-PARISIS	0.002545	0.500	3776.595745	3776.590735	0.005010
40847	PARIS 14	0.001259	0.250	9583.333333	9583.336603	0.003269

Entrée [20]:

```
import seaborn as sns
import matplotlib.pyplot as plt

D = final_predictions_train
D['y'] = y_train

#sns.pairplot(D,vars=D.columns[:-1], hue="y")
sns.pairplot(D,vars=D.columns)
plt.show()
```



4.1 - Mean error by City

Entrée [21]:

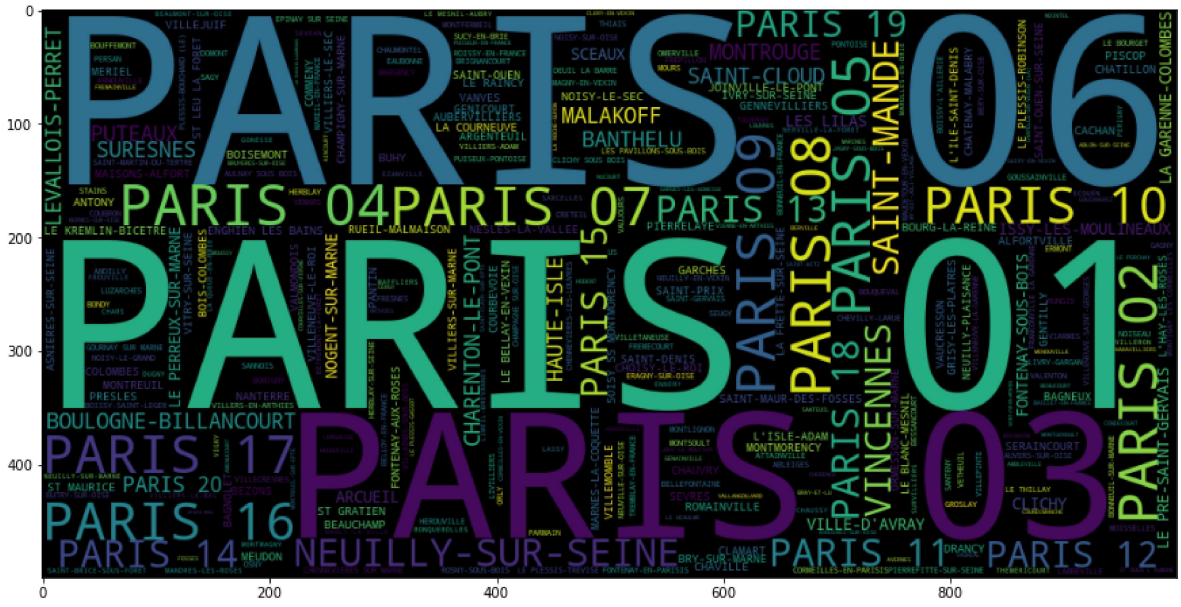
```
errors_city = final_predictions_train.groupby(['commune'])\
                        .agg({'error':['min','mean','max','std']})\
                        .reset_index()

errors_city.columns=['commune','error_min','error_mean','error_max','std_error']
errors_city=errors_city.sort_values(by=['error_mean'])
errors_city['inv_error_mean']=errors_city['error_mean'].apply(lambda v : 1/v)
errors_city['inv_std_error']=errors_city['std_error'].apply(lambda v : 1/v)
```

4.1.1 - City with high mean errors

Entrée [22]:

```
errors_city[['commune','error_mean']].tail(10)
ploter.show_cloud(errors_city)
```



4.1.2 - City with low mean errors

Entrée [23]:

```
errors_city[['commune','error_mean']].head(10)
ploter.show_cloud(errors_city,'inv_error_mean')
```

4.2 - Standard deviation of error by City

4.2.1 - City with high std errors

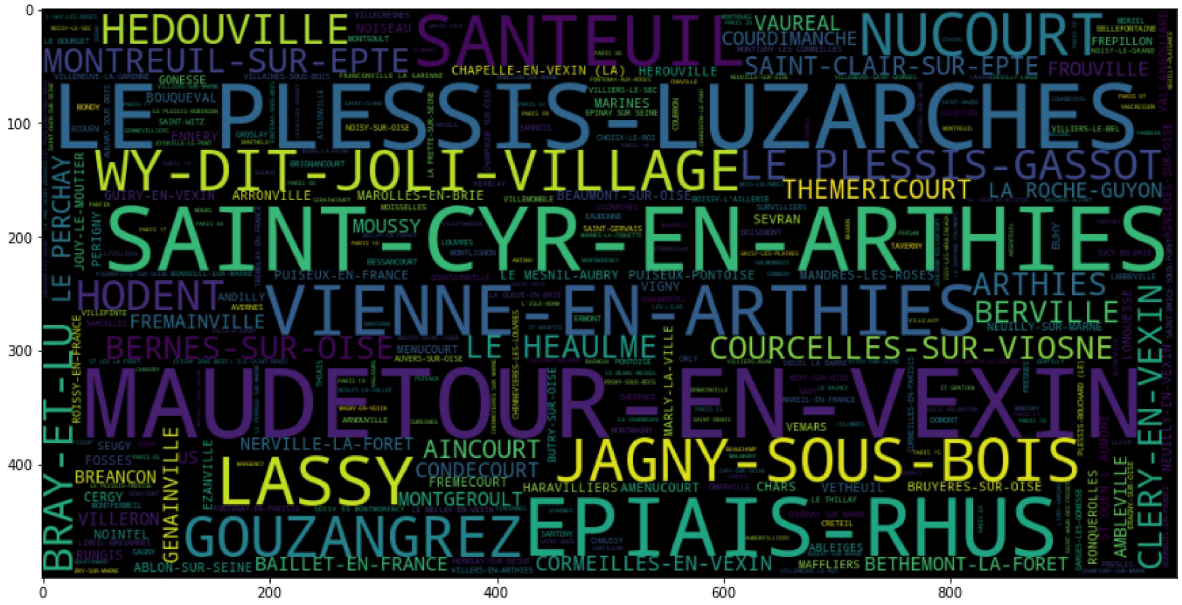
Entrée [24]:

```
ploter.show_cloud(errors_city.dropna(),'std_error')
```

4.2.2 - City with low std errors

Entrée [25]:

```
1 ploter.show_cloud(errors_city.dropna(),'inv_std_error')
```



5 - Conclusion

- The DVF dataset is useful for predicting future prices, as long as proper data preprocessing is applied.
- The encoding technique I applied definitely improves the performance on the Test set, even with a single encoded data. Nonetheless, the model is overfiting.
- Grid search should be applied to help improve model performance. But I didn't cover this part because the best model could change, depending on the area selected.
- Adding features specific to each home (the condition and practicality of the property) should help improve the model predictive power.
- The more we move away from Paris the less the model makes errors.
- It seems like the dynamic of the real estate market of Paris and nearby suburbs differs from that of distant suburbs. This needs to be investigated.

6 - Next step : Model deployment with Flask, Docker...