Anticiper les besoins en consommation de bâtiments

Présentation du projet

Seattle

ENERGY STAR

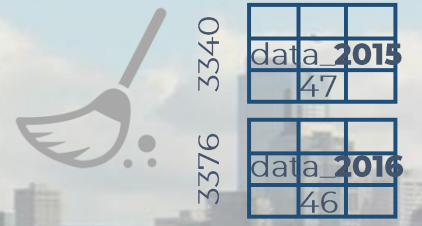
Nettoyage & Exploration des données

- Présentation du projet
- Chargement et paramétrages initiaux
- Nettoyage des données
- Feature engineering
- Synthèse étapes du nettoyage

Modélisation et évaluation

- Démarche suivie
- Chargement et jeu de test
- Jeu d'entrainement et Pipeline
- Modélisation avec GridSearchCV
- Evaluation des modèles
- Conclusion, apport EnergyStarScore

Chargement et paramétrages initiaux




```
#Modification des affichages de colonnes, lignes et largeurs de colonnes pour bien visualiser les colonnes
pd.set_option('display.max_columns', 200)
pd.set_option('display.max_rows', 20)
pd.set_option('display.max_colwidth', None)

#Affichage avec la bibliothèque graphique intégrée à Notebook:
%matplotlib inline

#Format des graphiques seaborn
sns.set_theme(style="whitegrid")
```

```
package : version
---------------------------
pandas  :  1.1.5
numpy   :  1.21.5
folium  :  0.12.1.post1
requests  :  2.27.1
seaborn  :  0.11.2
missingno  :  0.5.1
geopy  :  2.2.0
joblib  :  1.0.1
```

En 2015 on trouve 6 données différentes concaténées en une seule variable 'Location'.

```
data_2015['Location'].map(literal_eval).apply(pd.Series).head(3)
```

	latitude	longitude	human_address
0	47.61219025	-122.33799744	{"address": "405 OLIVE WAY", "city": "SEATTLE", "state": "WA", "zip": "98101"}
1	47.61310583	-122.33335756	{"address": "724 PINE ST", "city": "SEATTLE", "state": "WA", "zip": "98101"}
2	47.61334897	-122.33769944	{"address": "1900 5TH AVE", "city": "SEATTLE", "state": "WA", "zip": "98101"}

Cela fonctionne, latitude et longitude sont maintenant traités comme des colonnes à part. Nous allons pouvoir réaliser l'opération de split sur l'ensemble du jeu de données data_2015. Nous renouvelons l'opération sur le champ 'human adress' qui est lui aussi compacté.

```
data_2015= pd.concat([data_2015.drop(columns=['Location']),data_2015['Location']
                                .map(literal_eval).apply(pd.Series)],axis=1)
data_2015 = pd.concat([data_2015.drop(columns=['human_address']), data_2015['human_address']
                                .map(literal_eval).apply(pd.Series)], axis=1)
```

Notebook Jupyter est prêt pour le travail sur les données

Nettoyage des données

Normalisation 2015 / 2016 et merge

TaxParcelIdentificationNumber & ZipCode

Doublons | données récentes et uniques

Manquants par feature

Features numériques | consommations en %

Analyse qualitative | Bâtiments, Outliers
Features numériques | longitude, latitude
Analyse quantitative | Passage au log

Corrélations | Matrice, PairGrid, ANOVA
Encodage | PrimaryPropertyType

Etapes du nettoyage | graphique

Export pickle

```python
# fonction pour comparer les jeux de colonne entre deux DataFrames
# et qui renvoie les colonnes présentes uniquement dans chacun des deux df.
def column_comparator(df1,df2):
    columns_1 = list(df1.columns)
    columns_2 = list(df2.columns)
    diff_columns_1=[]
    diff_columns_2=[]

    for col in columns_2:
        if col in columns_1:
            continue
        else:
            diff_columns_2.append(col)
    for col in columns_1:
        if col not in columns_2:
            diff_columns_1.append(col)
    return diff_columns_1, diff_columns_2
```

```python
only_2015_columns, only_2016_columns = column_comparator(data_2015,data_2016)
```

```python
only_2015_columns
```

```
['OtherFuelUse(kBtu)',
 'GHGEmissions(MetricTonsCO2e)',
 'GHGEmissionsIntensity(kgCO2e/ft2)',
 'Comment',
 '2010 Census Tracts',
 'Seattle Police Department Micro Community Policing Plan Areas',
 'City Council Districts',
 'SPD Beats',
 'Zip Codes',
 'latitude',
 'longitude',
 'address',
 'city',
 'state',
 'zip']
```

≠

```python
only_2016_columns
```

```
['Address',
 'City',
 'State',
 'ZipCode',
 'Latitude',
 'Longitude',
 'Comments',
 'TotalGHGEmissions',
 'GHGEmissionsIntensity']
```

Normalisation 2015 / 2016 et merge

6716

45

1

```python
data_2015 = data_2015.rename(columns={"latitude":"Latitude", "longitude":"Longitude",
                                      "address":"Address", "city":"City",
                                      "state":"State", "zip":"ZipCode"})
```

2 On supprime les colonnes suivantes | peu de valeurs :

- OtherFuelUse(kBtu) | 17 lignes
- Comment | 13 lignes
- 2010 Census Tracts | 224 lignes

3 On renomme les colonnes suivantes :

```python
data_2015 = data_2015.rename(columns={"GHGEmissions(MetricTonsCO2e)":"TotalGHGEmissions"})
```

```python
data_2015 = data_2015.rename(columns={"GHGEmissionsIntensity(kgCO2e/ft2)":"GHGEmissionsIntensity"})
```

4 On supprime les colonnes suivantes | 2016 vs 2015:

- Zip Codes | données 2016 ≠ 2015 (incohérents)
- City Council Districts
- SPD Beats
- Seattle Police Department Micro Community Policing Plan Areas

```python
data = pd.concat([data_2015[data_2016.columns],data_2016],
                 axis = 0, copy=False).sort_values(["DataYear", "OSEBuildingID"])
data.shape
```

```
(6716, 45)
```

TaxParcelIdentificationNumber

```
data[data["TaxParcelIdentificationNumber"].str.len() > 10]
```

TaxParcelIdentificationNumber	CouncilDistrictCode	Neighborhood	Latitude
55696400000	3	CENTRAL	47.600609
6850700316 and 6850700315	3	EAST	47.622117
701100-0000	7	MAGNOLIA / QUEEN ANNE	47.625000

```
data["TaxParcelIdentificationNumber"] = data["TaxParcelIdentificationNumber"].str.replace("-","", regex=False)
```

```
data.loc[data["TaxParcelIdentificationNumber"].str.len() > 10,"TaxParcelIdentificationNumber"] =\
data.loc[data["TaxParcelIdentificationNumber"].str.len() > 10,"TaxParcelIdentificationNumber"].str[0:10]
```

```
data["TaxParcelIdentificationNumber"] = pd.to_numeric(data["TaxParcelIdentificationNumber"], downcast = 'integer')
```

ZipCode

6716

45

```
print("ZipCode contient-il des nan ?:",
      data.ZipCode.isnull().any(),"\n"\
      "Longitude contient-il des nan ?:",
      data.Longitude.isnull().any(),"\n"\
      "Latitude contient-il des nan ?:",\
      data.Latitude.isnull().any())
```

```
ZipCode contient-il des nan ?: True
Longitude contient-il des nan ?: False
Latitude contient-il des nan ?: False
```

```
geolocator = geopy.Nominatim(user_agent="study_project_Energy_Consumption_Forecast_Seattle")

def get_zip_code(x):
    location = geolocator.reverse("{}, {}".format(x['Latitude'],x['Longitude']), timeout=None)
    time.sleep(1)
    try:
        return location.raw['address']['postcode']
    except:
        pass
data['Zipcode_Calc'] = data.progress_apply(lambda x: get_zip_code(x), axis = 1)

data['Zipcode_Calc'].to_csv('Zipcode_Calc.csv')
```

```
In [296]: geolocator = geopy.Nominatim(user_agent="study_project_Energy_Consumption_Forecast_Seattle")

          def get_zip_code(x):
              location = geolocator.reverse("{}, {}".format(x['Latitude'],x['Longitude']), timeout=None)
              time.sleep(1)
              try:
                  return location.raw['address']['postcode']
              except:
                  pass
          data['Zipcode_Calc'] = data.progress_apply(lambda x: get_zip_code(x), axis = 1)

          Processing Dataframe: 100%|██████████| 6716/6716 [2:02:00<00:00,  1.09s/it]
```

```
# on charge le fichier csv 'Zipcode_Calc.csv' qui contient le résultat de la requête
data['Zipcode_Calc'] = pd.read_csv('Zipcode_Calc.csv', names=['Zipcode_Calc'])
```

Doublons | données récentes et uniques

3432

45

```
check_OSEB = data.loc[:,['OSEBuildingID', 'DataYear', 'BuildingType', 'TaxParcelIdentificationNumber']]
```

```
check_OSEB = check_OSEB.pivot_table(index=['OSEBuildingID', 'BuildingType'],\
                                    columns='DataYear', values='TaxParcelIdentificationNumber'\
                                    ).reset_index().rename_axis(None, axis=1)
```

```
test1 = check_OSEB[(check_OSEB[2015] != check_OSEB[2016]) & (check_OSEB[2015].notnull()) \
                   & (check_OSEB[2016].notnull())]['OSEBuildingID'].to_list()
```

```
pd.set_option('display.max_rows', 40)
data.loc[data['OSEBuildingID'].isin(test1),:].sort_values(by=['OSEBuildingID','DataYear'])
```

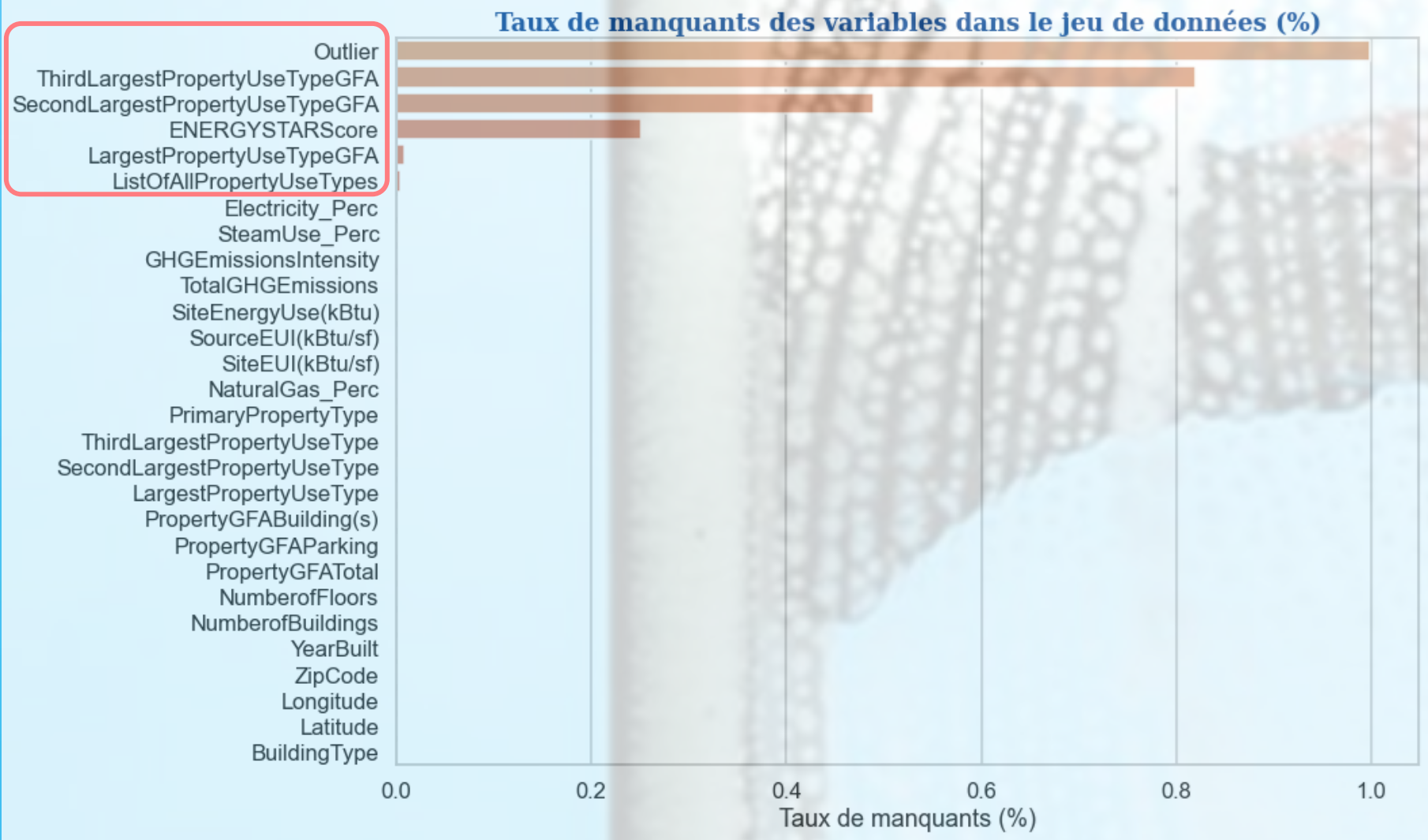
	OSEBuildingID	DataYear	BuildingType	PrimaryPropertyType	PropertyName	Address	City	State	TaxParcelIdentificationNumber	Latitude	Longitude	PropertyGFATotal
205	326	2015	NonResidential	Large Office	WEST LAKE TOWER OFFICE BUILDING	1601 5TH AVE	SEATTLE	WA	6.590000e+08	47.611753	-122.336674	369996
3543	326	2016	NonResidential	Large Office	Westlake Tower	1601 5th Avenue	Seattle	WA	9.301500e+09	47.612170	-122.336940	369996
1001	20432	2015	Multifamily MR (5-9)	Mid-Rise Multifamily	CAPCO PLAZA (Altamira Apartments & QFC)	4550 42ND AVE SW	SEATTLE	WA	9.520068e+08	47.561595	-122.385417	322592
4327	20432	2016	Multifamily MR (5-9)	Mid-Rise Multifamily	Altamira	4540 42nd Ave SW	Seattle	WA	1.333100e+09	47.561370	-122.385240	142099
3182	41928	2015	NonResidential	Supermarket/Grocery Store	SAFEWAY STORE #1586 (2012)	12318 15TH AVE NE	SEATTLE	WA	7.981010e+08	47.717930	-122.312202	51400
6481	41928	2016	NonResidential	Supermarket / Grocery Store	Safeway 1586 - 15th Ave	12318 15th Ave NE	Seattle	WA	6.798101e+09	47.718640	-122.312000	51400

```
#On trie sur la colonne 'DataYear' pour pouvoir ensuite conserver la dernière donnée saisie via keep='last'
data.sort_values(by=['DataYear'], ascending=True, inplace=True)
data.drop_duplicates(subset =['OSEBuildingID'], keep='last', inplace=True)
```

Manquants par feature

3267

28



Features numériques | consommations en %


```
columns_list = ['SteamUse(kBtu)', 'Electricity(kBtu)', 'NaturalGas(kBtu)']

# on somme les valeurs des colonnes de columns_list
data['VerifTotal'] = data[columns_list].sum(axis=1)
# on stocke dans EcartVerif la différence entre la conso SiteEnergyUse(kBtu) et VerifTotal
data['EcartVerif'] = data.apply(lambda x: x['SiteEnergyUse(kBtu)'] - x['VerifTotal'], axis=1)
# on calcule EcartVerif% en pourcentage du total réel:
data['EcartVerif%'] = data.apply(lambda x: round((x['EcartVerif'])/x['SiteEnergyUse(kBtu)'],2)*100 \
                                 if x['SiteEnergyUse(kBtu)'] != 0 else np.nan, axis=1)
```

```
data[(data['EcartVerif%']!=0) & (data['EcartVerif%'].notnull())]\
[['SiteEnergyUse(kBtu)', 'VerifTotal', 'EcartVerif', 'EcartVerif%']].shape
```

```
(43, 4)
```

```
data['SteamUse_Perc'] = data.apply(lambda x: x['SteamUse(kBtu)'] / x['SiteEnergyUse(kBtu)']\
                                   if x['SiteEnergyUse(kBtu)'] != 0 else np.nan, axis=1)
data['Electricity_Perc'] = data.apply(lambda x: x['Electricity(kBtu)'] / x['SiteEnergyUse(kBtu)']\
                                   if x['SiteEnergyUse(kBtu)'] != 0 else np.nan, axis=1)
data['NaturalGas_Perc'] = data.apply(lambda x: x['NaturalGas(kBtu)'] / x['SiteEnergyUse(kBtu)']\
                                   if x['SiteEnergyUse(kBtu)'] != 0 else np.nan, axis=1)
```

On arrondi à 4 décimales puis on multiplie par 100 pour obtenir un pourcentage ##,##% :

```
data['SteamUse_Perc'] = data.apply(lambda x: round((x['SteamUse_Perc']),4)*100, axis=1)
data['Electricity_Perc'] = data.apply(lambda x: round((x['SteamUse_Perc']),4)*100, axis=1)
data['NaturalGas_Perc'] = data.apply(lambda x: round((x['SteamUse_Perc']),4)*100, axis=1)
```

Analyse qualitative | Bâtiments

1552

28

```
print('La catégorie BuildingType comprend',len(data.BuildingType.unique()),\
      'valeurs uniques, pas de traitement particulier à appliquer')
```

```
La catégorie BuildingType comprend 8 valeurs uniques, pas de traitement particulier à appliquer
```

```
print('La catégorie comprend',len(data.PrimaryPropertyType.unique()),'valeurs, dont quelques doublons à corriger')
```

```
La catégorie comprend 31 valeurs, dont quelques doublons à corriger
```

Pour cela nous allons remplacer dans cette colonne les chaines de caractères :

- '\n' par une chaine vide
- ' / ' par '/'

```
data['PrimaryPropertyType'].replace(to_replace='\n$', value='', regex=True, inplace=True)
data['PrimaryPropertyType'].replace(to_replace=' / ', value='/', regex=True, inplace=True)
```

```
print('La catégorie PrimaryPropertyType comprend',len(data.PrimaryPropertyType.unique()),'valeurs uniques')
```

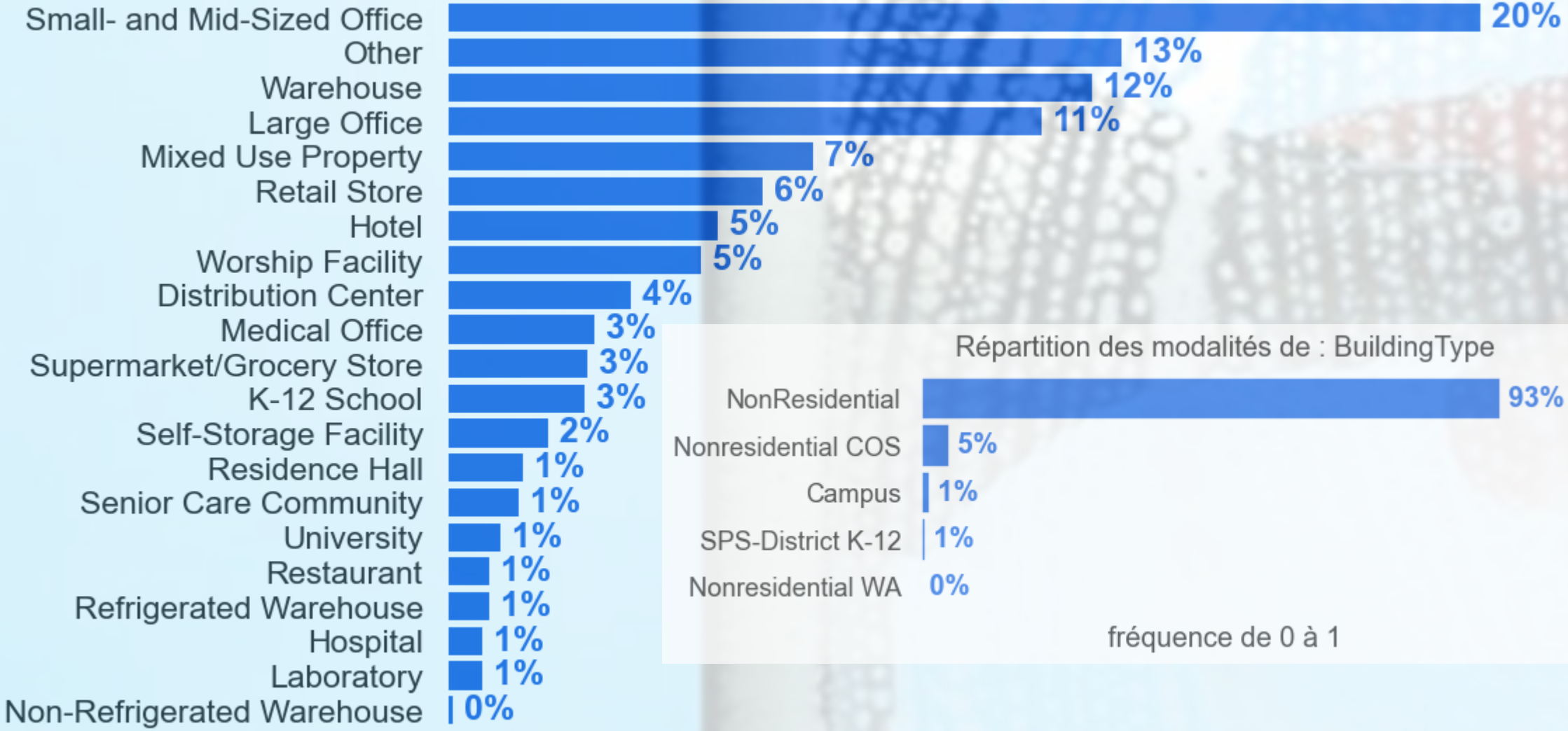
```
La catégorie PrimaryPropertyType comprend 27 valeurs uniques
```

```
data = data[(~data['BuildingType'].str.contains("Multifamily")) & (~data['PrimaryPropertyType'].str.contains("Multifamily"))]
```

Analyse qualitative | Bâtiments

1552

28



Analyse qualitative | Outliers

1550

27

```
data.loc[~data['Outlier'].isnull()]
```

gestPropertyUseType	ThirdLargestPropertyUseTypeGFA	ENERGYSTARScore	SiteEUI(kBtu/sf)	SourceEUI(kBtu/sf)	SiteEnergyUse(kBtu)	Outlier	TotalGHGEmissions
nan	0.0	NaN	6.2	19.5	133880.0	Low Outlier	0.93
nan	0.0	100.0	9.6	26.5	238255.0	Low Outlier	3.59

Features numériques | longitude, latitude

1549

25

```
# Fonction lambda de calcul de la distance geodesique au centre de Seattle

City_Center = (47.6062, -122.3321) # coordonnées gps du centre de Seattle

data['DistKm_CityCenter'] = data.apply(lambda x:geodesic((x["Latitude"], x["Longitude"]), City_Center), axis=1)
```

```
texte_test = '2.1922426617852855 km'
```

```
round(float(str(texte_test)[:-3]),3)
```

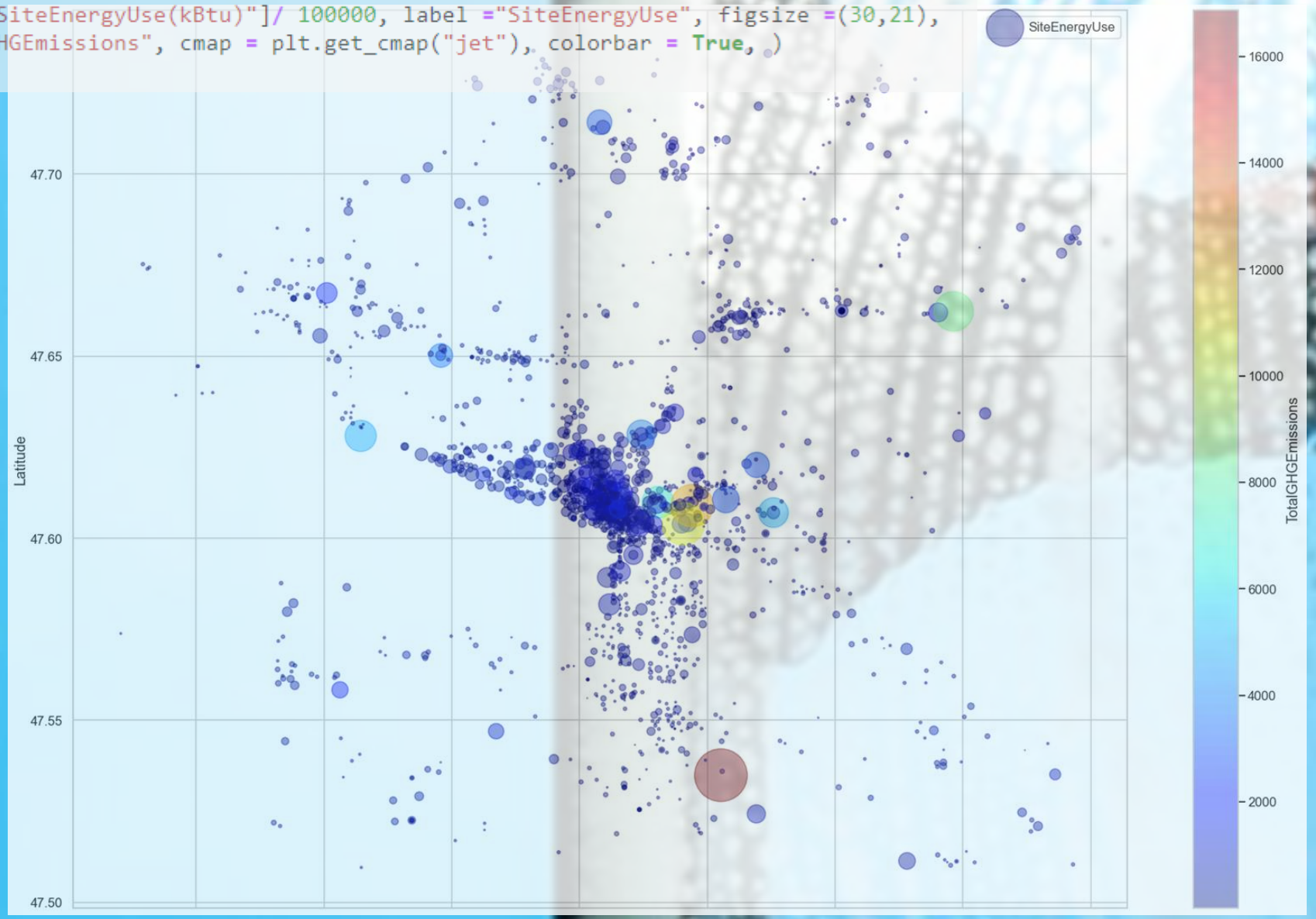
```
2.192
```

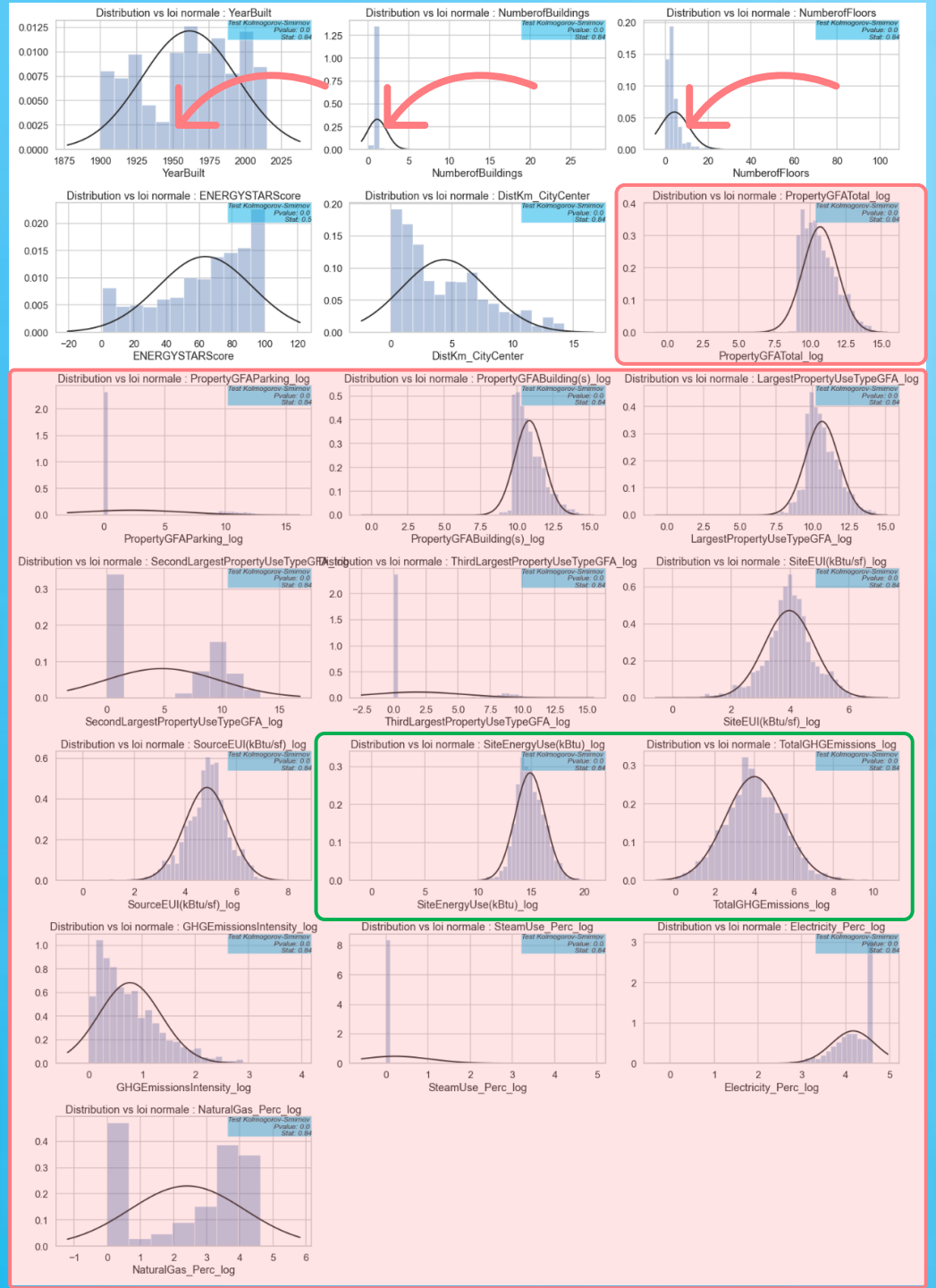
```
f = lambda x: round(float(str(x)[:-3]),3)
data['DistKm_CityCenter'] = data['DistKm_CityCenter'].apply(f)
```

```
data_num.append('DistKm_CityCenter')
```

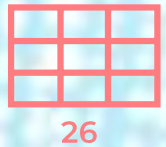
Features numériques | longitude, latitude


```
data.plot(kind ="scatter", x ="Longitude", y ="Latitude", alpha = 0.4,
            s = data["SiteEnergyUse(kBtu)"]/ 100000, label ="SiteEnergyUse", figsize =(30,21),
            c ="TotalGHGEmissions", cmap = plt.get_cmap("jet"), colorbar = True, )
plt.legend()
```





1548



26

Analyse quantitative | Passage au Log

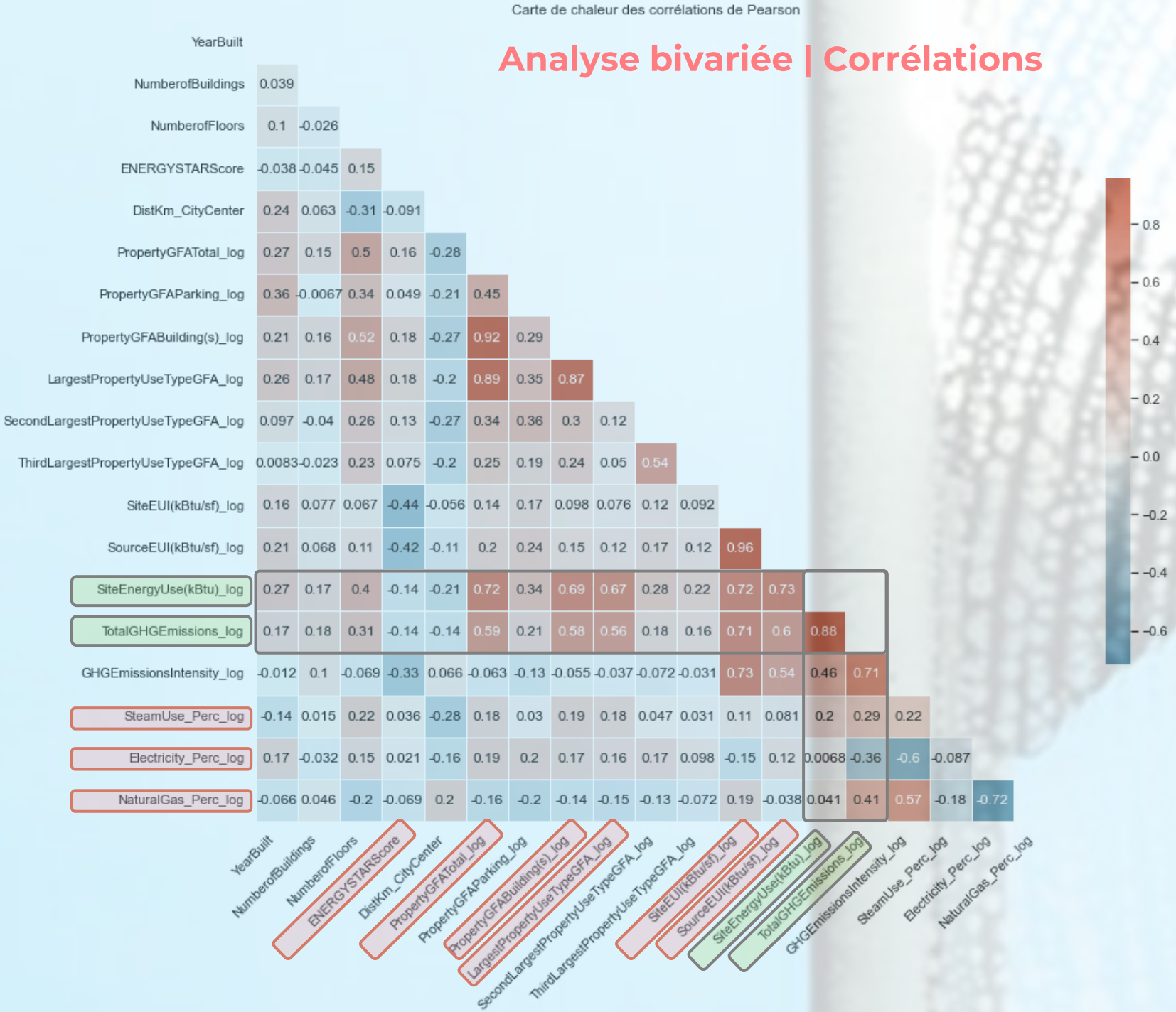
```
data_log_cols = ['PropertyGFATotal', 'PropertyGFAParking',
                 'PropertyGFABuilding(s)', 'LargestPropertyUseTypeGFA',
                 'SecondLargestPropertyUseTypeGFA', 'ThirdLargestPropertyUseTypeGFA',
                 'ENERGYSTARScore', 'SiteEUI(kBtu/sf)', 'SourceEUI(kBtu/sf)',
                 'SiteEnergyUse(kBtu)', 'TotalGHGEmissions', 'GHGEmissionsIntensity',
                 'SteamUse_Perc', 'Electricity_Perc', 'NaturalGas_Perc', 'DistKm_CityCenter']
```

```
# Fonction qui remplace par son logarithme chaque
# valeur non négative du jeu de colonnes spécifié
# (en ajoutant 1 afin de considérer 0).
# on conserve les colonnes originales dans data (pas de backup)

def switch_log(columns):
    # on passe au log les variables sélectionnées dans columns
    for column in columns:
        col_name = str(column + '_log')
        shift = 1 - data[column].min()
        data[col_name] = data[column].apply(lambda x: np.log(x + shift))
```

```
switch_log(data_log_cols)
```

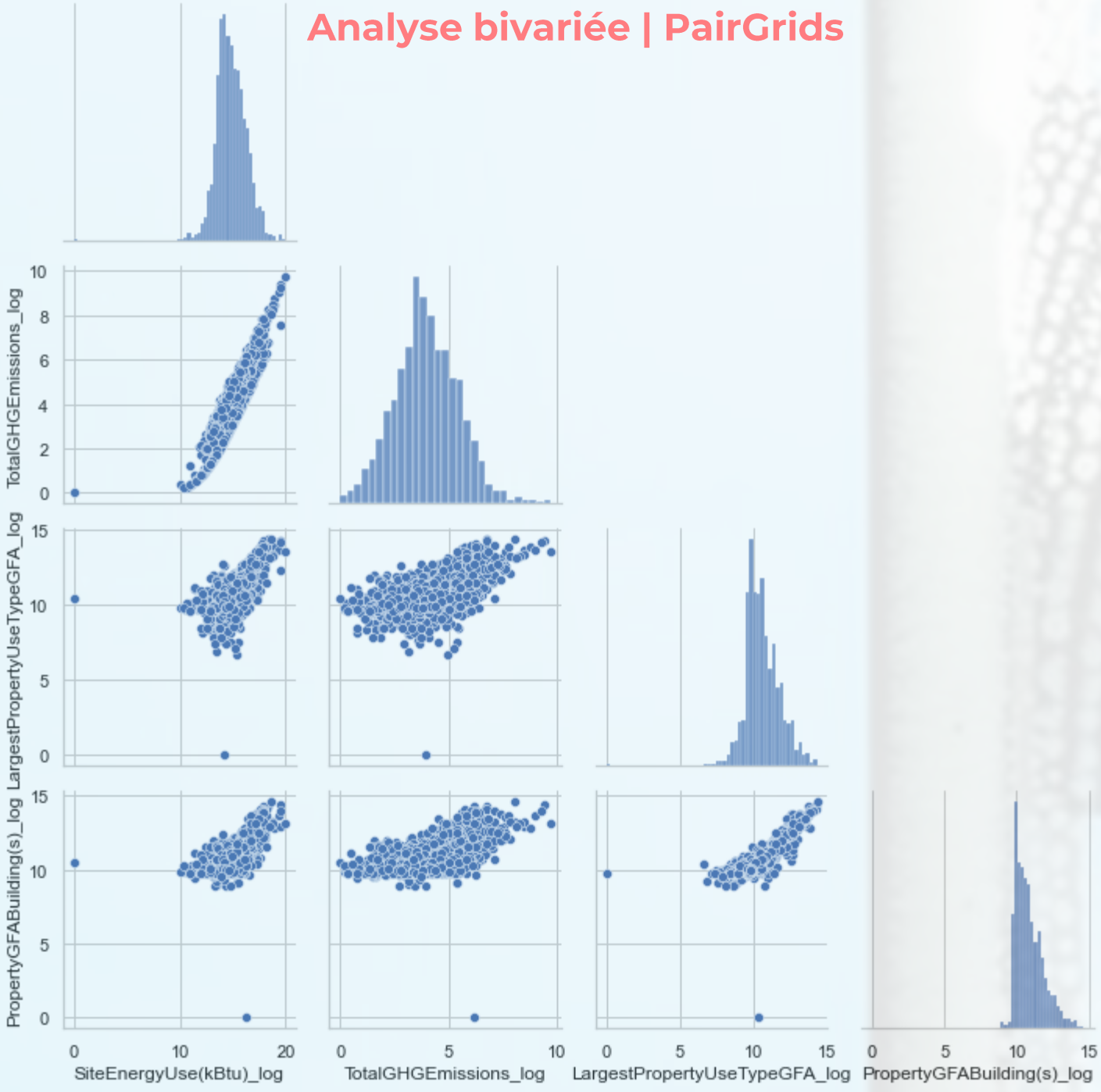
Analyse bivariée | Corrélations

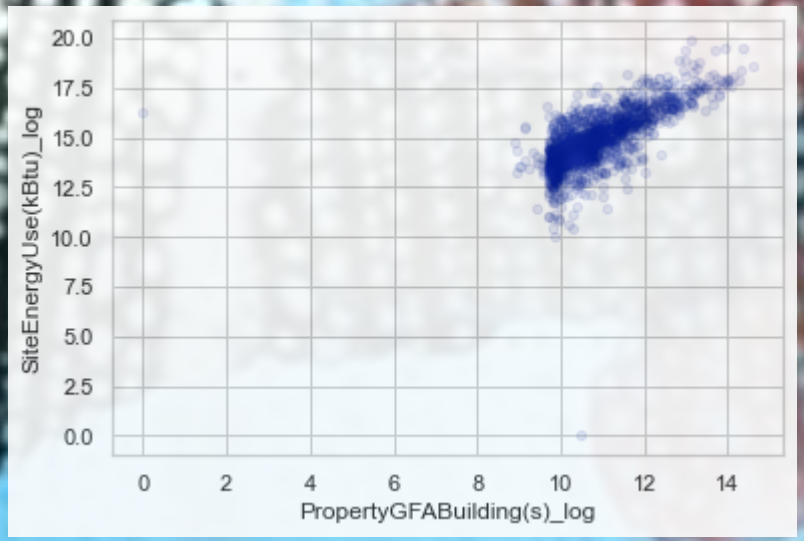


Corrélations > 0,7

	level_0	level_1	coef_correl
22	SourceEUI(kBtu/sf)_log	SiteEUI(kBtu/sf)_log	0.958613
20	PropertyGFATotal_log	PropertyGFABuilding(s)_log	0.924168
18	LargestPropertyUseTypeGFA_log	PropertyGFATotal_log	0.888886
16	SiteEnergyUse(kBtu)_log	TotalGHGEmissions_log	0.877741
14	PropertyGFABuilding(s)_log	LargestPropertyUseTypeGFA_log	0.869709
12	SourceEUI(kBtu/sf)_log	SiteEnergyUse(kBtu)_log	0.731697
10	GHGEmissionsIntensity_log	SiteEUI(kBtu/sf)_log	0.728344
8	SiteEnergyUse(kBtu)_log	SiteEUI(kBtu/sf)_log	0.721984
6	PropertyGFATotal_log	SiteEnergyUse(kBtu)_log	0.720223
4	TotalGHGEmissions_log	GHGEmissionsIntensity_log	0.714839
2	TotalGHGEmissions_log	SiteEUI(kBtu/sf)_log	0.708122
0	NaturalGas_Perc_log	Electricity_Perc_log	-0.718910

Analyse bivariée | PairGrids





Analyse bivariée | Anova multiples

	eta_carré
PrimaryPropertyType_vs_SiteEnergyUse(kBtu)_log	0.3950
PrimaryPropertyType_vs_TotalGHGEmissions_log	0.3311
LargestPropertyUseType_vs_SiteEnergyUse(kBtu)_log	0.3181
LargestPropertyUseType_vs_TotalGHGEmissions_log	0.2819
ZipCode_vs_SiteEnergyUse(kBtu)_log	0.1427
SecondLargestPropertyUseType_vs_SiteEnergyUse(kBtu)_log	0.1403
ZipCode_vs_TotalGHGEmissions_log	0.1150
ThirdLargestPropertyUseType_vs_SiteEnergyUse(kBtu)_log	0.0832
SecondLargestPropertyUseType_vs_TotalGHGEmissions_log	0.0805
ThirdLargestPropertyUseType_vs_TotalGHGEmissions_log	0.0589
BuildingType_vs_TotalGHGEmissions_log	0.0338
BuildingType_vs_SiteEnergyUse(kBtu)_log	0.0246


```python
# création de la fonction Anova (target, cat) qui va réaliser
# les Anova pour chaque variable cible et variable catégorielle
# et renvoyer les eta carrés correspondants

# On définit le rapport de corrélation eta carré pour l'afficher ensuite dans la fonction Anova
def eta_squared(x,y):
    moyenne_y = y.mean()
    classes = []
    for classe in x.unique():
        yi_classe = y[x==classe]
        classes.append({'ni': len(yi_classe),
                        'moyenne_classe': yi_classe.mean()})
    SCT = sum([(yj-moyenne_y)**2 for yj in y]) # somme des carrés totale
    SCE = sum([c['ni']*(c['moyenne_classe']-moyenne_y)**2 for c in classes]) # somme des carrés expliquée
    return SCE/SCT


list_vars_test = []
list_eta = []

def Anova(cat, target):
    print("Category_vs_Target : eta_carré")
    print("-"*50)
    for t in target:
        Y = t # quantitative target
        for c in cat:
            X = c # qualitative category
#           On crée le sous-échantillon (pas de filtre particulier ici)
            sous_echantillon = data[[X,Y]].dropna(how='any').copy()
            list_vars_test.append(c+"_vs_"+t)
            list_eta.append(eta_squared(sous_echantillon[X],sous_echantillon[Y]))

    rank_var = {}
    for i in range(len(list_vars_test)):
        print(list_vars_test[i]," : ",round(list_eta[i],4))
        rank_var.update({str(list_vars_test[i]):round(list_eta[i],4)})
        rank_var_df = pd.DataFrame.from_dict(rank_var, orient = 'index')
        rank_var_df.columns = ['eta_carré']
    return rank_var_df.sort_values(by='eta_carré',ascending=False)
```

Variables catégorielles | One Hot Encoding


```
nombre de modes par feature :
-----------------------------------
BuildingType : 5
PrimaryPropertyType : 21
LargestPropertyUseType : 58
SecondLargestPropertyUseType : 47
ThirdLargestPropertyUseType : 40
ZipCode : 44
```

```
PrimPropType = pd.get_dummies(data['PrimaryPropertyType'], prefix='PropType')
```

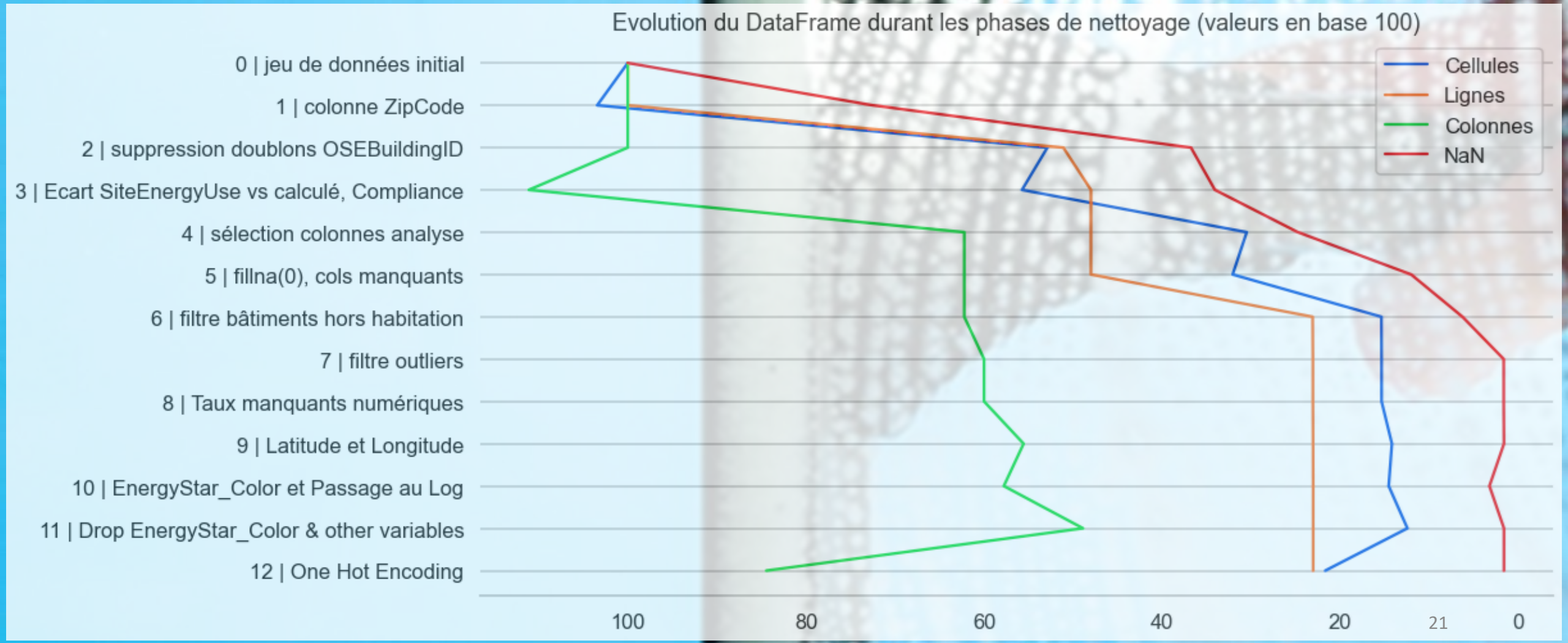
OSEBuildingID_DataYear	PropType_Distribution Center	PropType_Hospital	PropType_Hotel	PropType_K-12 School	PropType_Laboratory	PropType_Large Office	PropType_Medical Office	Pro
25042_2015	0	0	0	0	0	0	0	
24904_2015	0	0	0	0	0	0	0	
25455_2015	0	0	0	0	0	0	0	
25654_2015	0	0	0	0	0	0	0	
23163_2015	0	0	0	0	0	0	1	
...	...	...	...	...	...	...	...	
20985_2016	0	0	0	0	0	0	0	
20986_2016	0	0	0	0	0	0	0	
20987_2016	0	0	0	0	0	0	0	
20988_2016	0	0	0	0	0	0	0	
50226_2016	0	0	0	0	0	0	0	

1548 rows × 21 columns

```
# on concatène data et PrimPropType sur les colonnes
data = pd.concat([data,PrimPropType], axis = 1, copy=False)
```

```
data.drop(['BuildingType', 'PrimaryPropertyType', 'LargestPropertyUseType', 'SecondLargestPropertyUseType',\
           'ThirdLargestPropertyUseType'], axis=1, inplace=True)
```

Etapes de nettoyage | graphique



1548

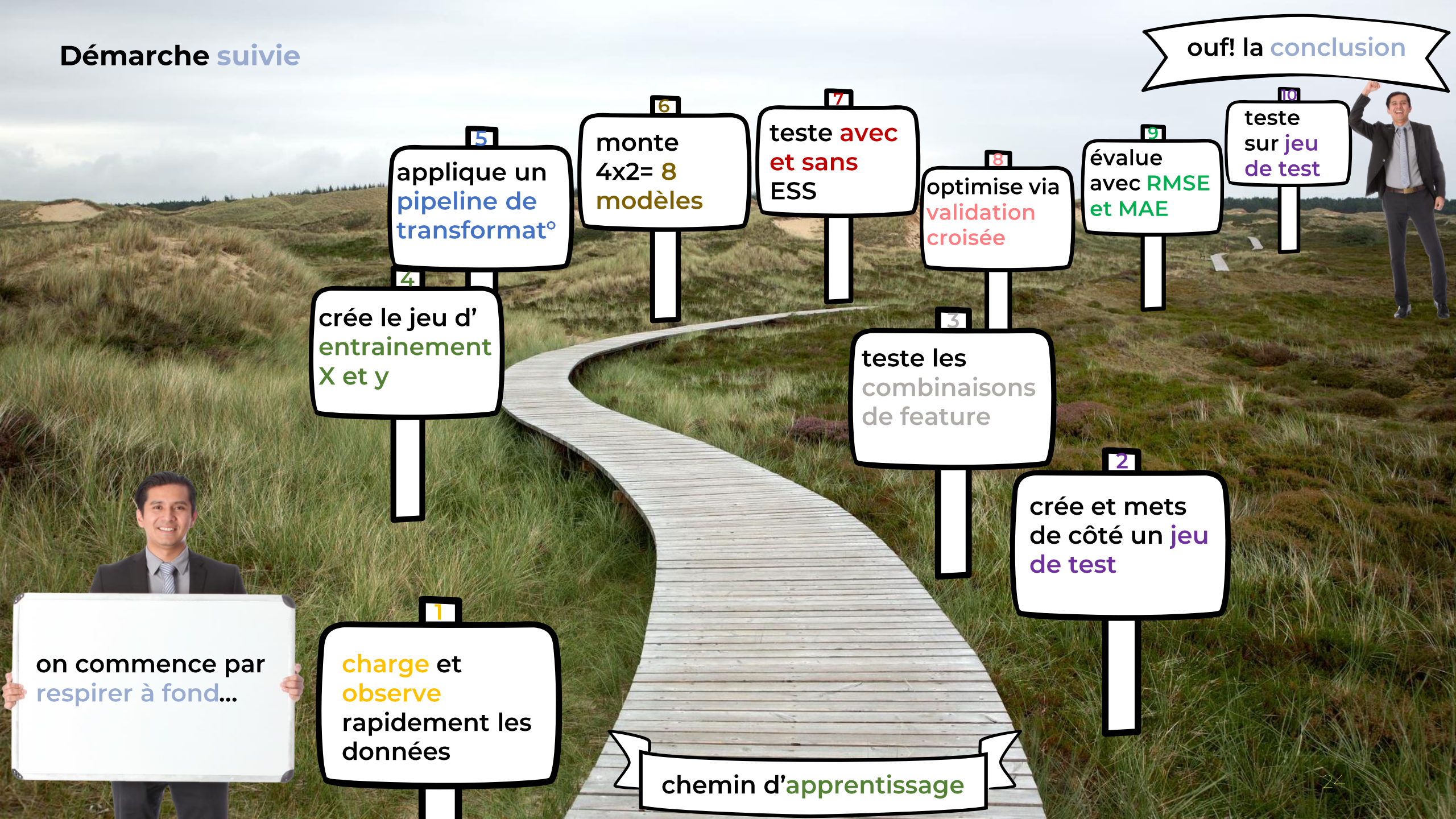
38


1548
38
Export pickle
data.to_pickle("data_cleaned.pkl")

Modélisation et évaluation

Démarche suivie
Chargement et jeu de test
Jeu d'entrainement et Pipeline
Modélisation avec GridSearchCV
Evaluation des modèles
Conclusion, apport EnergyStarScore

Démarche suivie



Chargement pickle et vérifications générales

```
def load_data():
    data = pd.read_pickle('data_cleaned.pkl')
    data_cat = pd.read_pickle('data_cat.pkl').values.tolist()
    data_num = pd.read_pickle('data_num.pkl').values.tolist()
    return (data, data_cat, data_num)
```

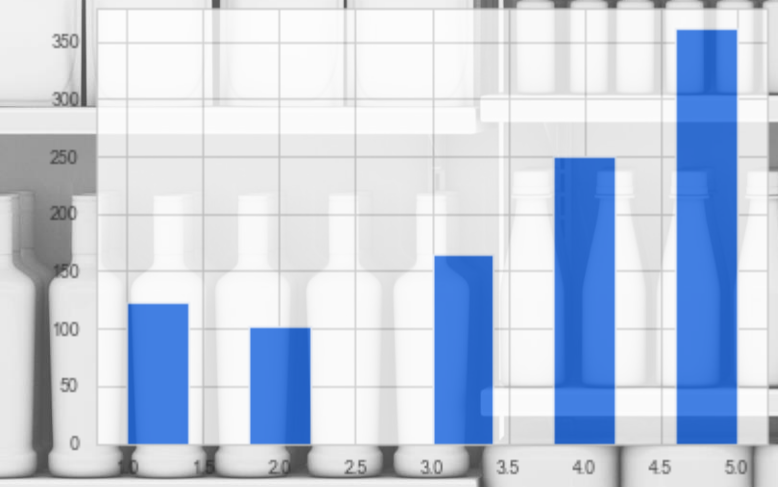
```
data, data_cat, data_num = load_data()
```

```
data.shape
```

```
(1548, 38)
```

Jeu train et test | attribut catégoriel EnergyStarScore_cat | échantillonnage stratifié

```python
# on supprimme les lignes avec nan de data (ENERGYSTARScore)
data.dropna(inplace=True)
# on réinitialise les indices
data.reset_index(drop=True, inplace=True)
# on découpe la variable en 5 classes étalées de 0 à 100 :
data["ENERGYSTARScore_cat"] = pd.cut(data["ENERGYSTARScore"],
                                     bins =[0, 20, 40, 60, 80, np.inf],
                                     labels =[1, 2, 3, 4, 5])
# on visualise la répartition par classe de la variable :
data["ENERGYSTARScore_cat"].hist()
```




```python
strat_test_set["ENERGYSTARScore_cat"].value_counts() / len(strat_test_set)
```

```
5    0.371084
4    0.245783
3    0.161446
1    0.118072
2    0.103614
Name: ENERGYSTARScore_cat, dtype: float64
```

```python
strat_train_set["ENERGYSTARScore_cat"].value_counts() / len(strat_train_set)
```

```
5    0.371377
4    0.245169
3    0.160628
1    0.118961
2    0.103865
Name: ENERGYSTARScore_cat, dtype: float64
```

```python
data["ENERGYSTARScore_cat"].value_counts() / len(data["ENERGYSTARScore_cat"])
```

```
5    0.371318
4    0.245292
3    0.160792
1    0.118783
2    0.103815
Name: ENERGYSTARScore_cat, dtype: float64
```

```python
# on crée les jeux de tets et d'entrainement avec la stratification selon "ENERGYSTAR_Score_cat"
from sklearn.model_selection import train_test_split
strat_train_set, strat_test_set = train_test_split(data, test_size = 0.2,
                                                   random_state = 42, stratify = data["ENERGYSTARScore_cat"])
```

33 Features utilisés pour la modélisation

Numériques

NumberofBuildings
NumberofFloors
ENERGYSTARScore
PropertyGFATotal_log
PropertyGFAParking_log
PropertyGFABuilding(s)_log
LargestPropertyUseTypeGFA_log
SecondLargestPropertyUseTypeGFA_log
ThirdLargestPropertyUseTypeGFA_log
SteamUse_Perc_log
Electricity_Perc_log
NaturalGas_Perc_log

Catégoriels encodés en numériques

PropType_Distribution Center
PropType_Hospital
PropType_Hotel
PropType_K-12 School
PropType_Laboratory
PropType_Large Office
PropType_Medical Office
PropType_Mixed Use Property
PropType_Non-Refrigerated Warehouse
PropType_Other
PropType_Refrigerated Warehouse
PropType_Residence Hall
PropType_Restaurant
PropType_Retail Store
PropType_Self-Storage Facility
PropType_Senior Care Community
PropType_Small- and Mid-Sized Office
PropType_Supermarket/Grocery Store
PropType_University
PropType_Warehouse
PropType_Worship Facility

Pipeline de Transformations | Simple Imputer et StandardScaler

feature scaling / mise à l'échelle
StandardScaler

Imputation par la médiane
SimpleImputer

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
num_pipeline = Pipeline([('imputer', SimpleImputer(strategy ="median")),
                         ('std_scaler', StandardScaler())])
# créons un num_pipeline_rfo sans le StandardScaler
# pour RandomForest (pas de std scaling)
num_pipeline_rfo = Pipeline([('imputer', SimpleImputer(strategy ="median"))])
```

RandomForest = pas de StandardScaler

```
from sklearn.compose import ColumnTransformer

num_attribs = X_num
# cat_attribs = data_cat
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs)])

X_prepared = full_pipeline.fit_transform(X)
```

```
# créons un jeu de données préparées pour RandomForest,
# en passant par ColumnTransformer
num_attribs = X_num
# cat_attribs = data_cat

full_pipeline_rfo = ColumnTransformer([
    ("num", num_pipeline_rfo, num_attribs)])

X_prepared_rfo = full_pipeline_rfo.fit_transform(X)
```

Construction des 4 modèles | hyperparamètres via validation croisée

Dummy (baseline)

```
from sklearn.dummy import DummyRegressor
dum_reg = DummyRegressor(strategy="mean")
dum_reg.fit(X_prepared, y["SiteEnergyUse(kBtu)_log"])
```

Linear

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_prepared, y["SiteEnergyUse(kBtu)_log"])
```

Ridge

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge

param_grid = [{'alpha': [1, 10, 100, 1000],
               'random_state': [20220519]},
              ]

ridge_reg = Ridge()

grid_search = GridSearchCV(ridge_reg, param_grid, cv = 5,
                           scoring ='neg_mean_squared_error',
                           return_train_score = True)

grid_search.fit(X_prepared, y["SiteEnergyUse(kBtu)_log"])
```

SVM

```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

param_grid = {'C': [0.1, 1, 10, 100, 1000, 5000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

svm_reg = svm.SVR()

grid_search = GridSearchCV(svm_reg, param_grid, cv = 5,
                           scoring ='neg_mean_squared_error',
                           return_train_score = True)

grid_search.fit(X_prepared, y["SiteEnergyUse(kBtu)_log"])
```

Random Forest

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

param_grid = [{'max_features': [0.2, 0.3, 0.4, 0.5, 0.8],
               # number or perc of features considered for
               # splitting at each leaf node
               'n_estimators': [1000],# number of trees in the forest
               'random_state': [20220519]}]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv = 5,
                           scoring ='neg_mean_squared_error',
                           return_train_score = True)

grid_search.fit(X_prepared_rfo, y["SiteEnergyUse(kBtu)_log"])
```

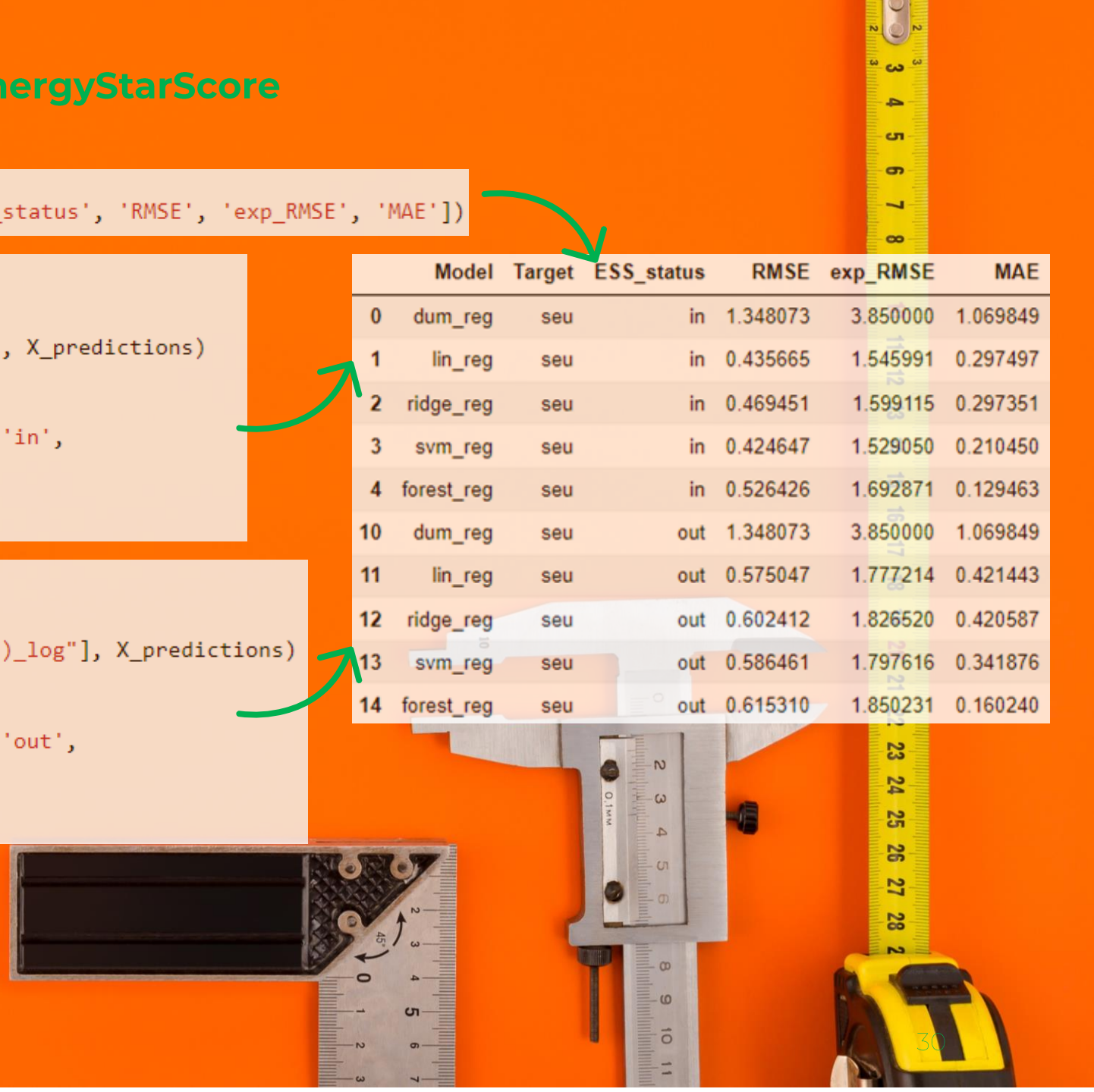
Evaluation des modèles | avec et sans EnergyStarScore

```
# on crée un DataFrame pour recueillir les évaluations
base_results = pd.DataFrame(columns=['Model', 'Target', 'ESS_status', 'RMSE', 'exp_RMSE', 'MAE'])
```

```
forest_reg_seu = grid_search.best_estimator_
X_predictions = forest_reg_seu.predict(X_prepared_rfo)
forest_mae = mean_absolute_error(y["SiteEnergyUse(kBtu)_log"], X_predictions)
# on charge et récupère le score du modèle
base_results = base_results.append(
    {'Model' : 'forest_reg', 'Target': 'seu', 'ESS_status' : 'in',
     'RMSE' : np.sqrt(-grid_search.best_score_),
     'exp_RMSE' : exp(np.sqrt(-grid_search.best_score_)),
    'MAE' : forest_mae}, ignore_index=True)
```

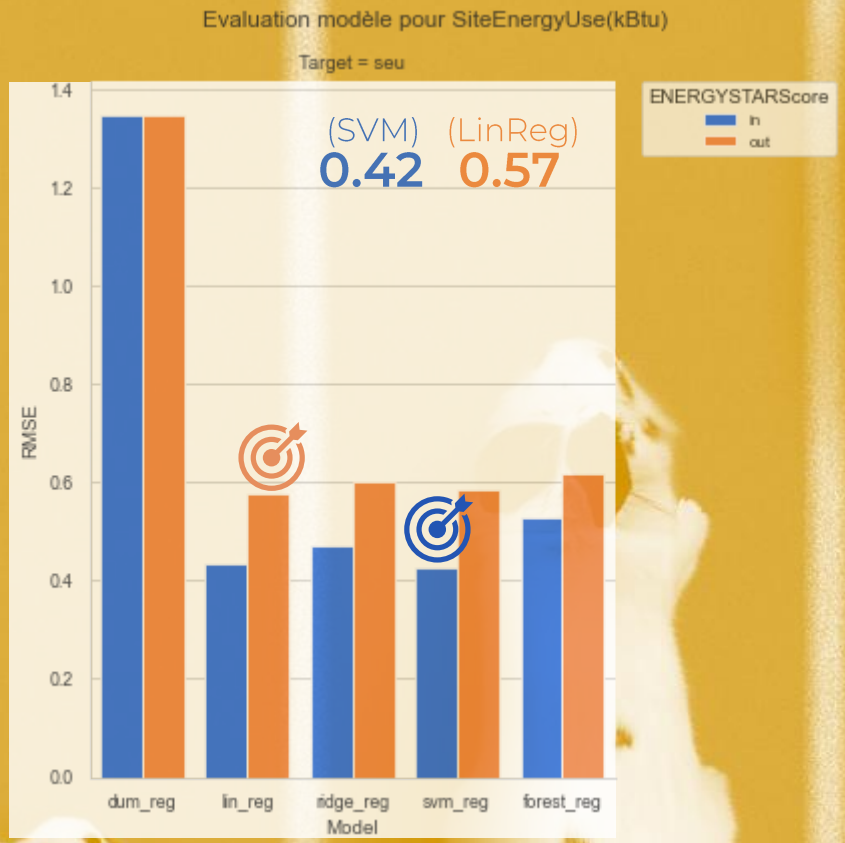
```
best_reg_seu_wo_ess = grid_search.best_estimator_
X_predictions = best_reg_seu_wo_ess.predict(X_prepared_rfo)
forest_mae_wo_ess = mean_absolute_error(y["SiteEnergyUse(kBtu)_log"], X_predictions)
# on charge et récupère le score du modèle
base_results = base_results.append(
    {'Model' : 'forest_reg', 'Target': 'seu', 'ESS_status' : 'out',
     'RMSE' : np.sqrt(-grid_search.best_score_),
     'exp_RMSE' : exp(np.sqrt(-grid_search.best_score_)),
    'MAE' : forest_mae_wo_ess}, ignore_index=True)
```

	Model	Target	ESS_status	RMSE	exp_RMSE	MAE
0	dum_reg	seu	in	1.348073	3.850000	1.069849
1	lin_reg	seu	in	0.435665	1.545991	0.297497
2	ridge_reg	seu	in	0.469451	1.599115	0.297351
3	svm_reg	seu	in	0.424647	1.529050	0.210450
4	forest_reg	seu	in	0.526426	1.692871	0.129463
10	dum_reg	seu	out	1.348073	3.850000	1.069849
11	lin_reg	seu	out	0.575047	1.777214	0.421443
12	ridge_reg	seu	out	0.602412	1.826520	0.420587
13	svm_reg	seu	out	0.586461	1.797616	0.341876
14	forest_reg	seu	out	0.615310	1.850231	0.160240

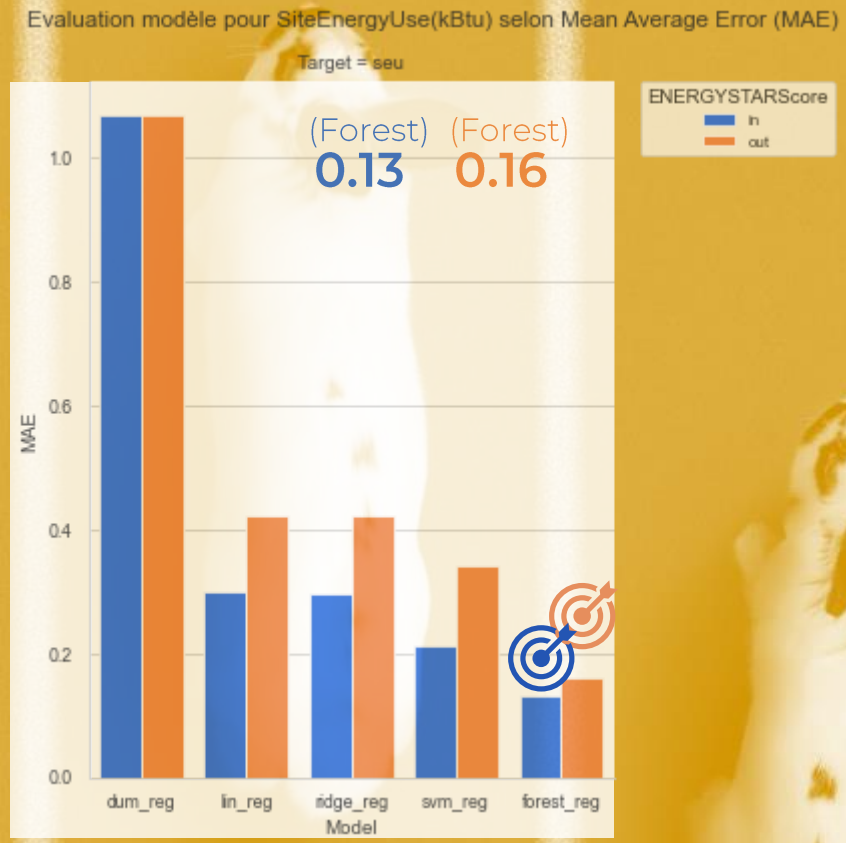


Comparaison des erreurs RMSE et MAE | SiteEnergyUse(kBtu)_log

RMSE

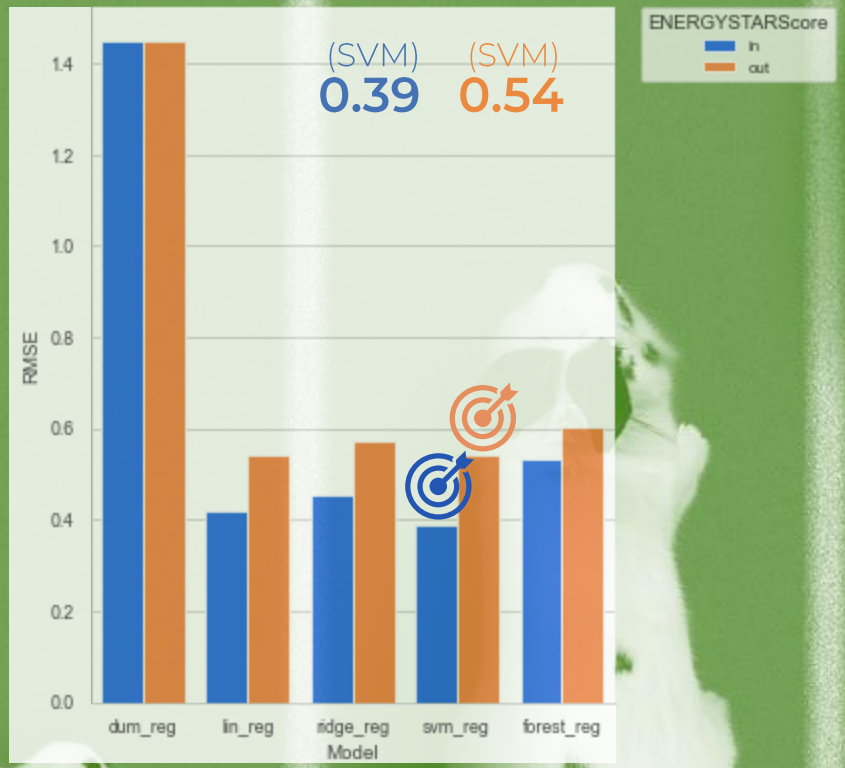


MAE

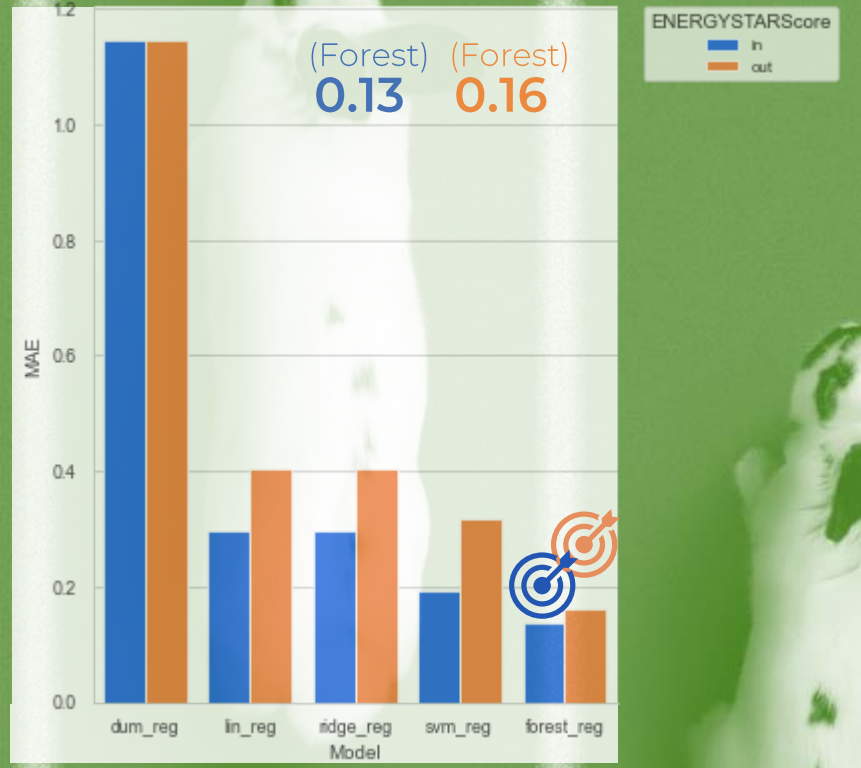


Comparaison des erreurs RMSE et MAE | TotalGHGEmissions_log

RMSE

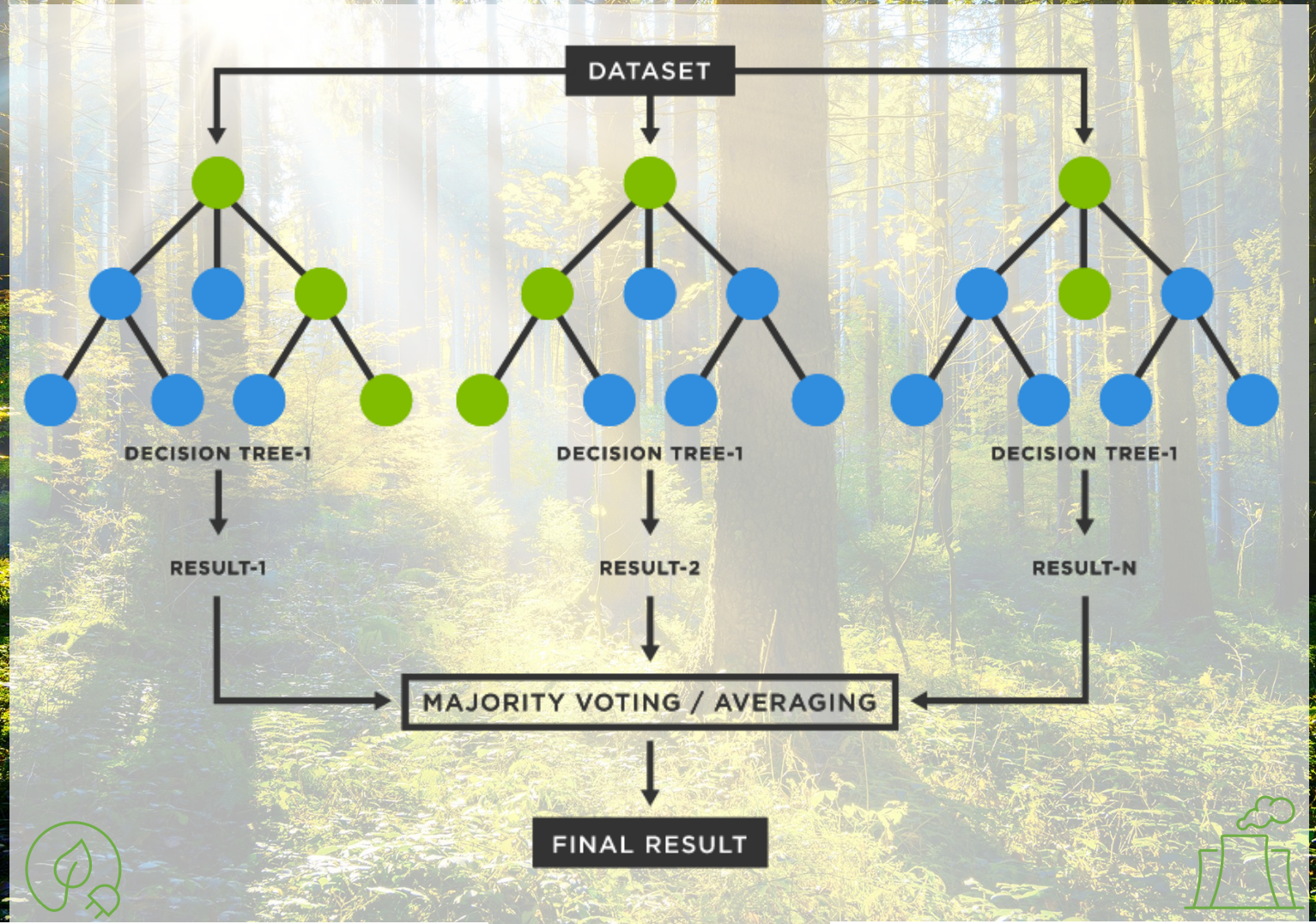


MAE




1
Random
Forest
(MAE)

Présentation modèle Random Forest



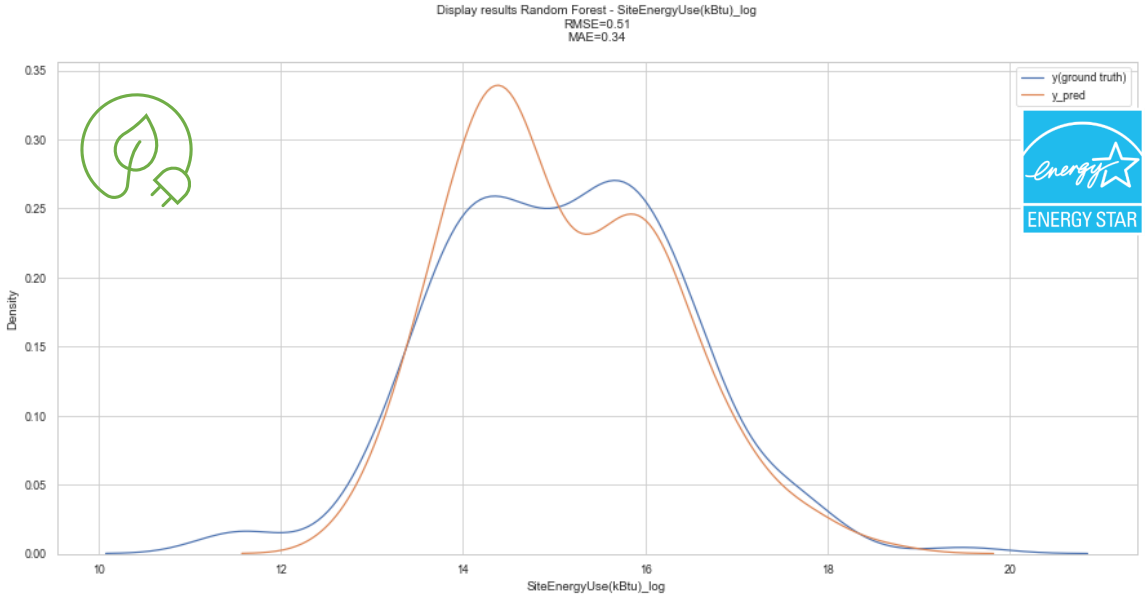

```
RandomForestRegressor(max_features=0.3, n_estimators=1000,
                      random_state=20220519)
```

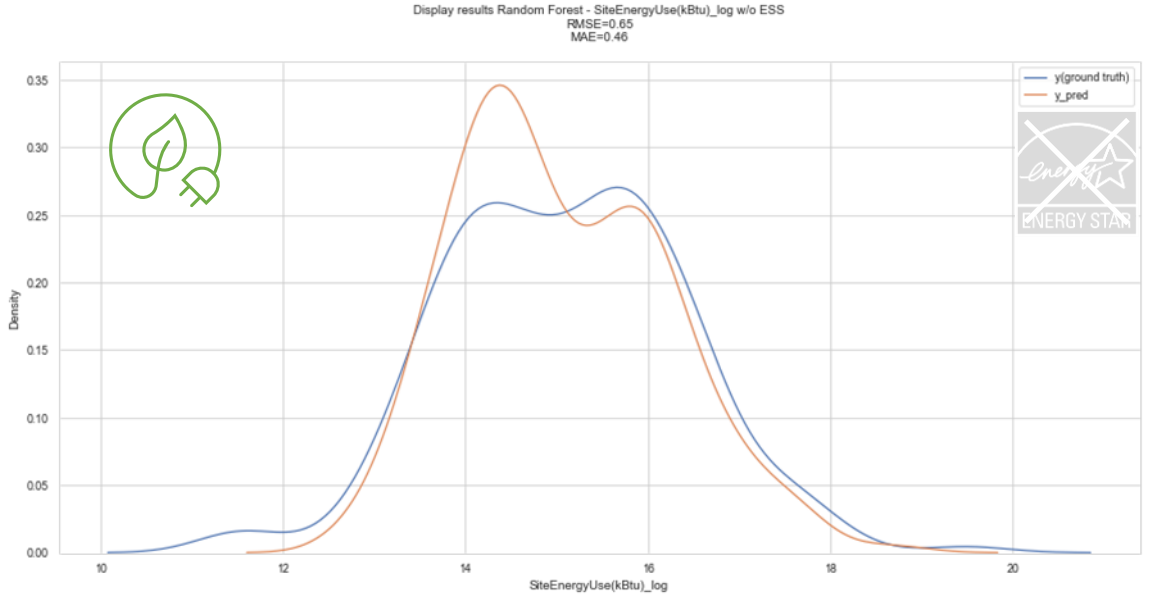
```
RandomForestRegressor(max_features=0.2, n_estimators=1000,
                      random_state=20220519)
```

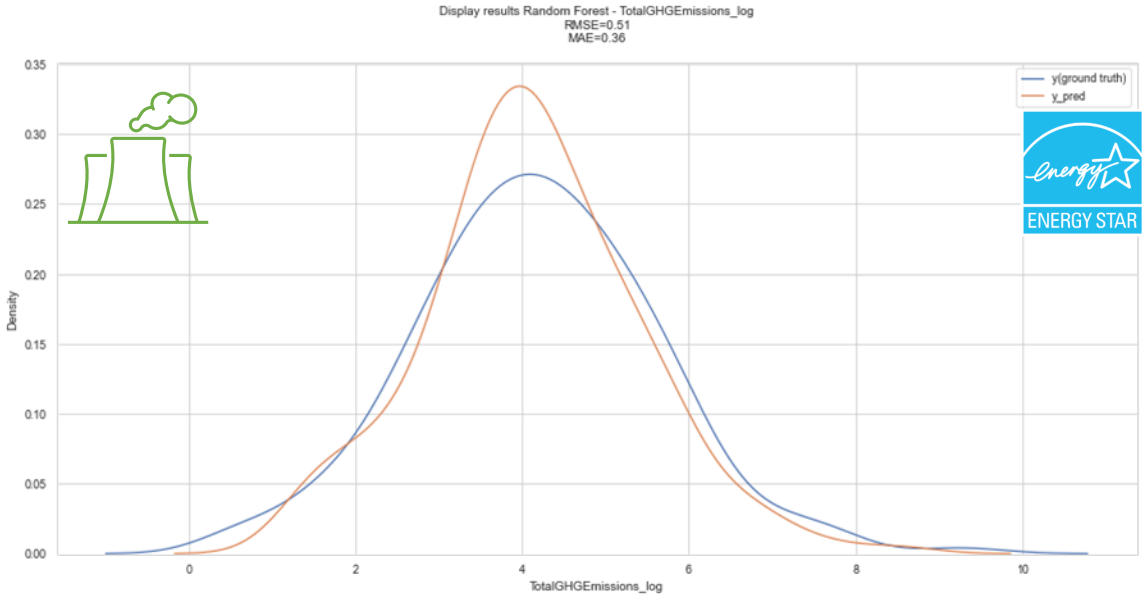
Evaluation RMSE avec Random Forest sur jeu de test | avec et sans EnergyStarScore

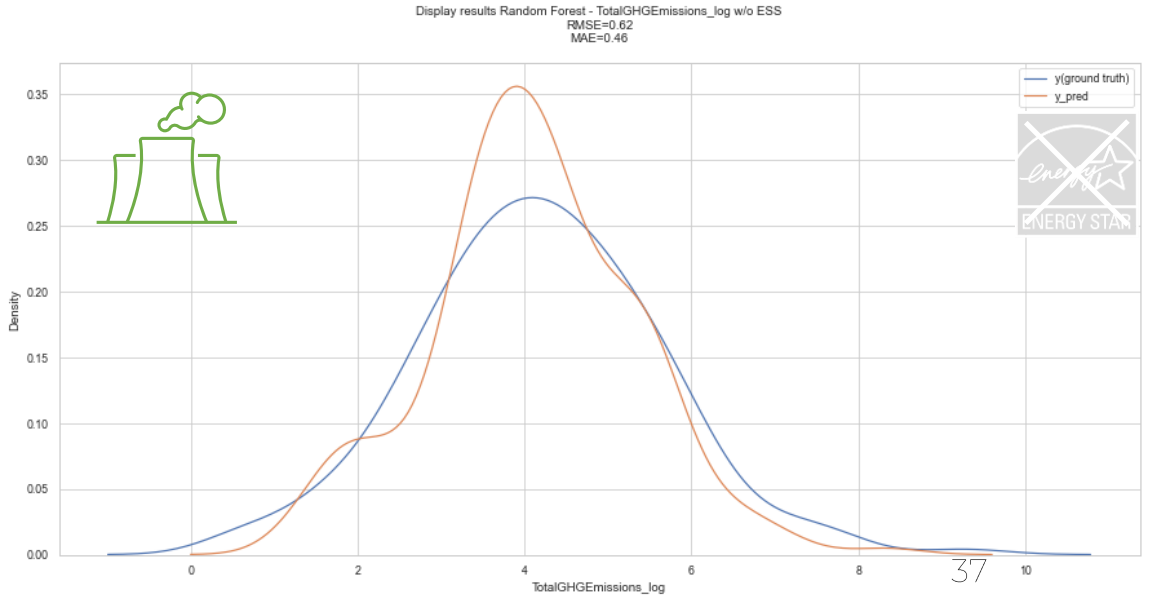
Evaluation MAE avec Random Forest sur jeu de test | avec et sans EnergyStarScore

Erreur du modèle Random Forest selon l'estimation par noyau | kernel density estimation










Questions ?


Merci !