Anticipez les besoins en consommations de bâtiments

Elodie Chen

Projet 3

Soutenance du 07/02/2026

Contexte

Dans le cadre de l'objectif de **neutralité carbone** à l'horizon 2050, la ville de Seattle s'intéresse à la **consommation énergétique** et aux **émissions de CO_2** des bâtiments *non destinés à l'habitation*.

Des relevés détaillés ont été réalisés par les services municipaux en 2016. À partir de ces données, l'objectif de ce projet est de développer des modèles capables de prédire la **consommation totale d'énergie** et les **émissions de CO_2** de **bâtiments non résidentiels** pour lesquels ces informations ne sont pas encore disponibles.

Présentation des données & EDA

```
1 building_consumption.sample(10)
```

	OSEBuildingID	DataYear	BuildingType	PrimaryPropertyType	PropertyName	Address	City	State	ZipCode	TaxParcelIdentificationNumber	CouncilE
1670	23064	2016	NonResidential	Other	Evergreen Washelli	11111 Aurora Ave. N	Seattle	WA	98133.0	3026049008	
614	820	2016	SPS-District K-12	K-12 School	Bailey Gatzert Elementary	1301 E. Yesler Way	Seattle	WA	98122.0	0007600137	
2401	25531	2016	Multifamily MR (5-9)	Mid-Rise Multifamily	Waterside	1550 Alki Ave SW	Seattle	WA	98103.0	9197900000	
2745	26825	2016	Multifamily LR (1-4)	Low-Rise Multifamily	leschi heights	2900 s king st	Seattle	WA	98144.0	9552200165	
864	20052	2016	NonResidential	Retail Store	WEST BLDG	4629 26th Avenue NE	Seattle	WA	98105.0	0925049346	
61	86	2016	NonResidential	Hotel	SCCA - ALLIANCE HOUSE	207 Pontius Ave N	Seattle	WA	98109.0	2467400430	

CouncilDistrictCode	Neighborhood	Latitude	Longitude	YearBuilt	NumberofBuildings	NumberofFloors	PropertyGFATotal	PropertyGFAParking
5	NORTHWEST	47.70969	-122.34557	1970	5.0	1	23166	0
3	CENTRAL	47.60120	-122.31548	1988	1.0	1	52924	0
1	SOUTHWEST	47.58946	-122.39376	1994	1.0	6	44188	0
3	CENTRAL	47.59849	-122.29465	1967	1.0	2	24621	0
4	NORTHEAST	47.66246	-122.29898	1997	1.0	1	63153	0
7	LAKE UNION	47.61988	-122.33211	2008	1.0	6	84103	20732

Dimensions:
3376 lignes, 46 colonnes

```
# On regarde le nombre de valeurs manquantes par colonne ainsi que leur type
building_consumption.info()
```

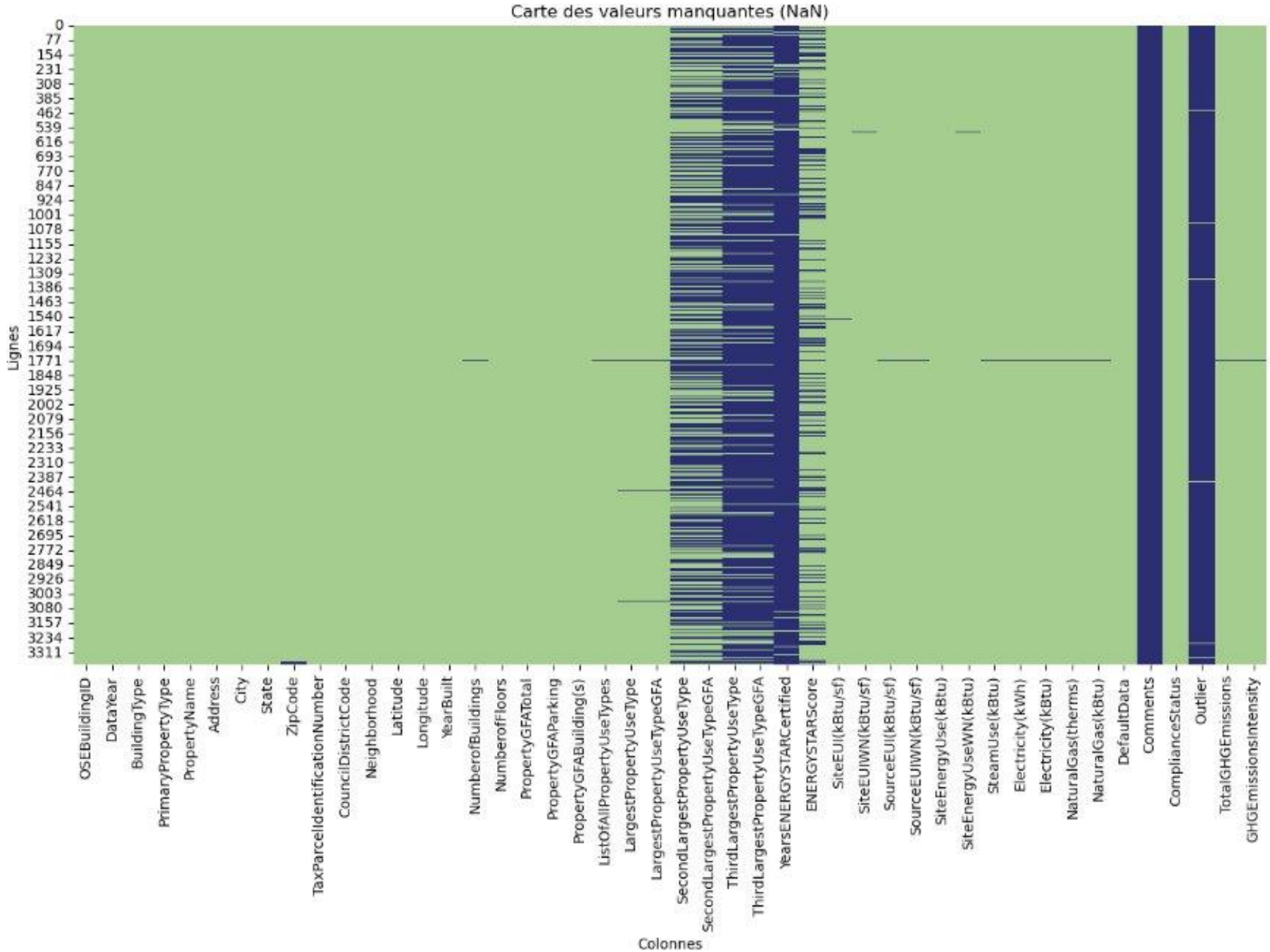
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3376 entries, 0 to 3375
Data columns (total 46 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   OSEBuildingID                    3376 non-null   int64
 1   DataYear                         3376 non-null   int64
 2   BuildingType                     3376 non-null   object
 3   PrimaryPropertyType              3376 non-null   object
 4   PropertyName                     3376 non-null   object
 5   Address                          3376 non-null   object
 6   City                             3376 non-null   object
 7   State                            3376 non-null   object
 8   ZipCode                          3360 non-null   float64
 9   TaxParcelIdentificationNumber    3376 non-null   object
 10  CouncilDistrictCode              3376 non-null   int64
 11  Neighborhood                     3376 non-null   object
 12  Latitude                         3376 non-null   float64
 13  Longitude                        3376 non-null   float64
 14  YearBuilt                        3376 non-null   int64
 15  NumberofBuildings                3368 non-null   float64
 16  NumberofFloors                   3376 non-null   int64
 17  PropertyGFATotal                 3376 non-null   int64
 18  PropertyGFAParking               3376 non-null   int64
 19  PropertyGFABuilding(s)           3376 non-null   int64
 20  ListOfAllPropertyUseTypes        3367 non-null   object
 21  LargestPropertyUseType           3356 non-null   object
 22  LargestPropertyUseTypeGFA        3356 non-null   float64
 23  SecondLargestPropertyUseType     1679 non-null   object
 24  SecondLargestPropertyUseTypeGFA  1679 non-null   float64
 25  ThirdLargestPropertyUseType      596 non-null    object
 26  ThirdLargestPropertyUseTypeGFA   596 non-null    float64
 27  YearsENERGYSTARCertified         119 non-null    object
 28  ENERGYSTARScore                  2533 non-null   float64
 29  SiteEUI(kBtu/sf)                 3369 non-null   float64
 30  SiteEUIWN(kBtu/sf)               3370 non-null   float64
 31  SourceEUI(kBtu/sf)               3367 non-null   float64
 32  SourceEUIWN(kBtu/sf)             3367 non-null   float64
 33  SiteEnergyUse(kBtu)              3371 non-null   float64
 34  SiteEnergyUseWN(kBtu)            3370 non-null   float64
 35  SteamUse(kBtu)                   3367 non-null   float64
 36  Electricity(kWh)                 3367 non-null   float64
 37  Electricity(kBtu)                3367 non-null   float64
 38  NaturalGas(therms)               3367 non-null   float64
 39  NaturalGas(kBtu)                 3367 non-null   float64
 40  DefaultData                      3376 non-null   bool
 41  Comments                         0 non-null      float64
 42  ComplianceStatus                 3376 non-null   object
 43  Outlier                          32 non-null     object
 44  TotalGHGEmissions                3367 non-null   float64
 45  GHGEmissionsIntensity            3367 non-null   float64
dtypes: bool(1), float64(22), int64(8), object(15)
memory usage: 1.2+ MB
```

I. 2. b) Première identification des valeurs manquantes

```
building_consumption.isnull().sum()
```

```
OSEBuildingID                       0
DataYear                            0
BuildingType                        0
PrimaryPropertyType                 0
PropertyName                        0
Address                             0
City                                0
State                               0
ZipCode                            16
TaxParcelIdentificationNumber       0
CouncilDistrictCode                 0
Neighborhood                        0
Latitude                            0
Longitude                           0
YearBuilt                           0
NumberofBuildings                   8
NumberofFloors                      0
PropertyGFATotal                    0
PropertyGFAParking                  0
PropertyGFABuilding(s)              0
ListOfAllPropertyUseTypes           9
LargestPropertyUseType             20
LargestPropertyUseTypeGFA          20
SecondLargestPropertyUseType     1697
SecondLargestPropertyUseTypeGFA  1697
ThirdLargestPropertyUseType      2780
ThirdLargestPropertyUseTypeGFA   2780
YearsENERGYSTARCertified         3257
ENERGYSTARScore                   843
SiteEUI(kBtu/sf)                    7
SiteEUIWN(kBtu/sf)                  6
SourceEUI(kBtu/sf)                  9
SourceEUIWN(kBtu/sf)                9
SiteEnergyUse(kBtu)                 5
SiteEnergyUseWN(kBtu)               6
SteamUse(kBtu)                      9
Electricity(kWh)                    9
Electricity(kBtu)                   9
NaturalGas(therms)                  9
NaturalGas(kBtu)                    9
DefaultData                         0
Comments                         3376
ComplianceStatus                    0
Outlier                          3344
TotalGHGEmissions                   9
GHGEmissionsIntensity               9
dtype: int64
```

Dimensions:
3376 lignes, 46 colonnes


Carte des valeurs manquantes (NaN)
Lignes
0
77
154
231
308
385
462
539
616
693
770
847
924
1001
1078
1155
1232
1309
1386
1463
1540
1617
1694
1771
1848
1925
2002
2079
2156
2233
2310
2387
2464
2541
2618
2695
2772
2849
2926
3003
3080
3157
3234
3311
OSEBuildingID
DataYear
BuildingType
PrimaryPropertyType
PropertyName
Address
City
State
ZipCode
TaxParcelIdentificationNumber
CouncilDistrictCode
Neighborhood
Latitude
Longitude
YearBuilt
NumberofBuildings
NumberofFloors
PropertyGFATotal
PropertyGFAParking
PropertyGFABuilding(s)
ListOfAllPropertyUseTypes
LargestPropertyUseType
LargestPropertyUseTypeGFA
SecondLargestPropertyUseType
SecondLargestPropertyUseTypeGFA
ThirdLargestPropertyUseType
ThirdLargestPropertyUseTypeGFA
YearsENERGYSTARCertified
ENERGYSTARScore
SiteEUI(kBtu/sf)
SiteEUIWN(kBtu/sf)
SourceEUI(kBtu/sf)
SourceEUIWN(kBtu/sf)
SiteEnergyUse(kBtu)
SiteEnergyUseWN(kBtu)
SteamUse(kBtu)
Electricity(kWh)
Electricity(kBtu)
NaturalGas(therms)
NaturalGas(kBtu)
DefaultData
Comments
ComplianceStatus
Outlier
TotalGHGEmissions
GHGEmissionsIntensity
Colonnes

- Filtrage des bâtiments non résidentiels

```
df = building_consumption[
building_consumption['BuildingType'].str.contains('NonResidential', na=False)
].copy()
```

```
Dimensions avant filtrage: (3376, 46)

Dimensions après filtrage: (1460, 46)
```

- Suppression des données redondantes ou n'apportant pas d'information supplémentaires

```
print(df['Outlier'].unique())
print(df['Comments'].unique())
print(df['BuildingType'].unique())
print(df['City'].unique())
print(df['State'].unique())
```

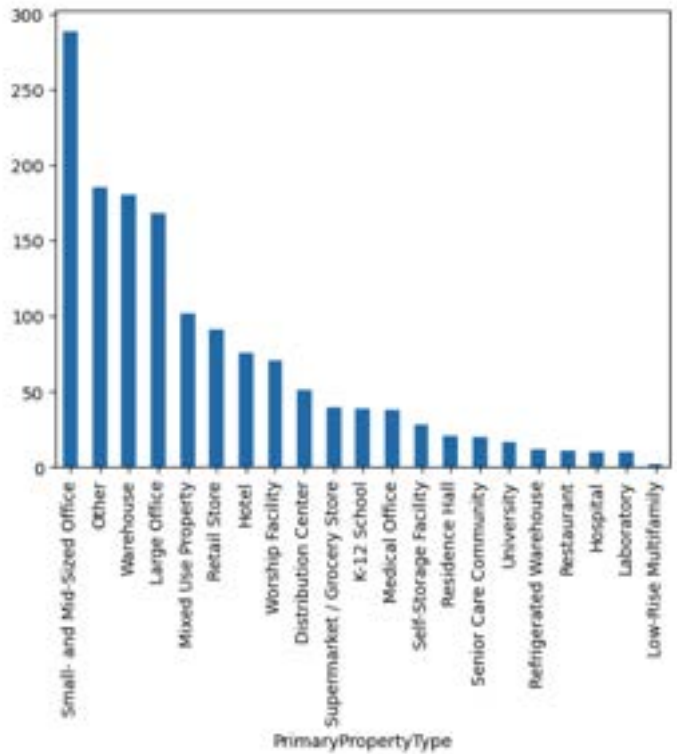
```
[nan 'High outlier' 'Low outlier']
[nan]
['NonResidential']
['Seattle']
['WA']
```

La réciproque se vérifie également : on compte exactement 113 lignes pour lesquelles la colonne ['ComplianceStatus'] vaut 'Error - Correct Default Data', soit précisément le même nombre de lignes où ['DefaultData'] est **True**.

On peut donc en conclure que l'information portée par ['DefaultData'] (**True**/**False**) est déjà entièrement contenue dans ['ComplianceStatus'], notamment à travers la valeur 'Error - Correct Default Data'.

```
1 df['PrimaryPropertyType'].value_counts().plot(kind='bar')
```

```
<Axes: xlabel='PrimaryPropertyType'>
```

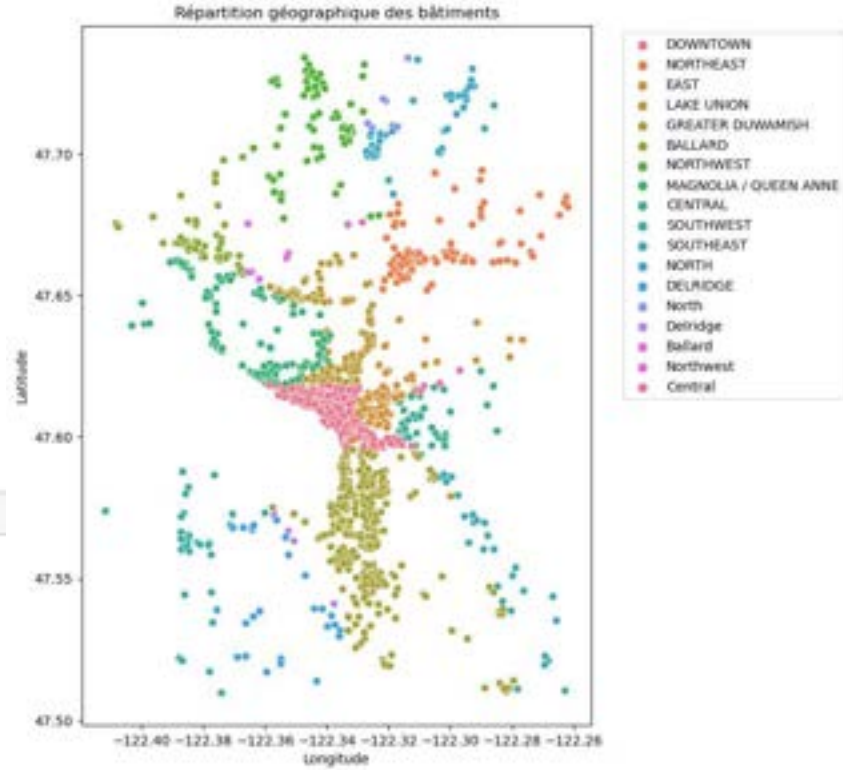



```
1 df.loc[~df['YearsENERGYSTARCertified'].isna(), 'YearsENERGYSTARCertified'].unique()
```

```
array(['2016', '2014', '2012', '20172015',
       '201720162015201420132012201120102020092008', '201020092007',
       '201720102007', '20162015', '201520142012201120092008',
       '20162015201420132011201020092007200', '201620092008',
       '201620142012', '20162015201420122008',
       '2016201520142013201220112010200920082007200620052004',
       '201520142013', '20162015201420132013', '20172016201520142013201120112008',
       '20172016201520142013200', '2016201520132012201',
       '2015201320122010', '20162011201020092008', '2007', '2010',
       '20172016201520092008', '20172013200920008', '2015', '2017',
       '20172011201020007', '20172016201420132012201120102020092007',
       '20172015201420122010200', '20092005', '20172015201420132',
       '2016201520122', '20122010200720006', '20112009', '2009',
       '20162015201420132009200', '20162015201420132011209',
       '20162015201420132012201120102', '20172016201220', '20102008',
       '20162014201220112008200', '20172015201420132011200920008',
       '20162015201420132012201020092', '20162011201020092',
       '20162015201220', '20162012', '20162013201220112010200920008',
       '20172016201520122009200720006', '20162015201320122011201020092008',
       '20142013201220008',
       '2016201520142013201220112010200920082007200620052003200220000',
       '20142010200820011', '2011', '20172015201420132011', '20172016',
       '2013', '20152012', '201720162015', '201620152014',
       '20162015201420132012201020092008', '201620152010200920008'],
      dtype=object)
```

```
1 df['ListOfAllPropertyUseTypes']
```

```
0                                                   Hotel
1                                 Hotel, Parking, Restaurant
2                                                   Hotel
3                                                   Hotel
4                                 Hotel, Parking, Swimming Pool
                                  ...
3338                                Non-Refrigerated Warehouse
3339                                                Office
3340                                      Other - Recreation
3347                                     K-12 School, Parking
3356    Data Center, Laboratory, Museum, Office, Other...
Name: ListOfAllPropertyUseTypes, Length: 1460, dtype: object
```



Nettoyage et mise en forme

II. 1. Préparation au feature engineering

II. 1. a) Traitement du contenu de la colonne 'ListOfAllPropertyUseTypes'

- Création de colonnes plus facilement exploitables : 'parsed', 'parsed_simple', 'parsed_simple_grouped'

Création d'une fonction pour séparer les éléments d'une chaîne en utilisant uniquement les virgules qui se trouvent en dehors des parenthèses

```
# Test pour vérifier la fonction créée
split_outside_parentheses("Other - Mall, Personal Services (Health/Beauty, Dry Cleaning, etc)")
```

```
['Other - Mall', 'Personal Services (Health/Beauty, Dry Cleaning, etc)']
```

Création d'une fonction qui ne garde que les parties non contenues dans des parenthèses

```
# Tests pour vérifier la fonction de simplification

simplify("Other - Mall, Personal Services (Health/Beauty, Dry Cleaning, etc)") # simplifié car commence par 'Other'
```

0.0s

```
'Other'
```

```
simplify("Hospital (General Medical & Surgical), Parking") # simplifié car comporte une ouverture de parenthèse '('
```

0.0s

```
'Hospital'
```

```
df['ListOfAllPropertyUseTypes']
```

```
0                                                  Hotel
1                                Hotel, Parking, Restaurant
2                                                  Hotel
3                                                  Hotel
4                              Hotel, Parking, Swimming Pool
                                    ...
3338                            Non-Refrigerated Warehouse
3339                                                Office
3340                                     Other - Recreation
3347                                   K-12 School, Parking
3356    Data Center, Laboratory, Museum, Office, Other...
Name: ListOfAllPropertyUseTypes, Length: 1460, dtype: object
```

```
# Affichage des types d'usage ainsi que leur compte
with pd.option_context('display.max_rows', None, 'display.max_rows', None)
    display(df['parsed_simple'].explode().value_counts())
```

```
parsed_simple
Office                                692
Parking                               406
Other                                 281
Retail Store                          250
Non-Refrigerated Warehouse            245
Restaurant                             98
Hotel                                  81
Worship Facility                       76
Distribution Center                    61
Medical Office                         61
Supermarket/Grocery Store              54
K-12 School                            46
Multifamily Housing                    37
Data Center                            36
Self-Storage Facility                  35
Residence Hall/Dormitory               23
Laboratory                             22
Senior Care Community                  20
Bank Branch                            19
College/University                     18
Social/Meeting Hall                    17
Financial Office                       17
Refrigerated Warehouse                 16
```

```
parsed_simple_grouped
Other                          786
Office                         692
Parking                        406
Retail Store                   250
Non-Refrigerated Warehouse     245
Restaurant                      98
Hotel                           81
Worship Facility                76
Medical Office                  61
Distribution Center             61
Name: count, dtype: int64
```

```
Vocational School                     2
Convention Center                     1
Vocat                                 1
Single Family Home                    1
Wholesale Club/Supercenter            1
Enclosed Mall                         1
Name: count, dtype: int64
```

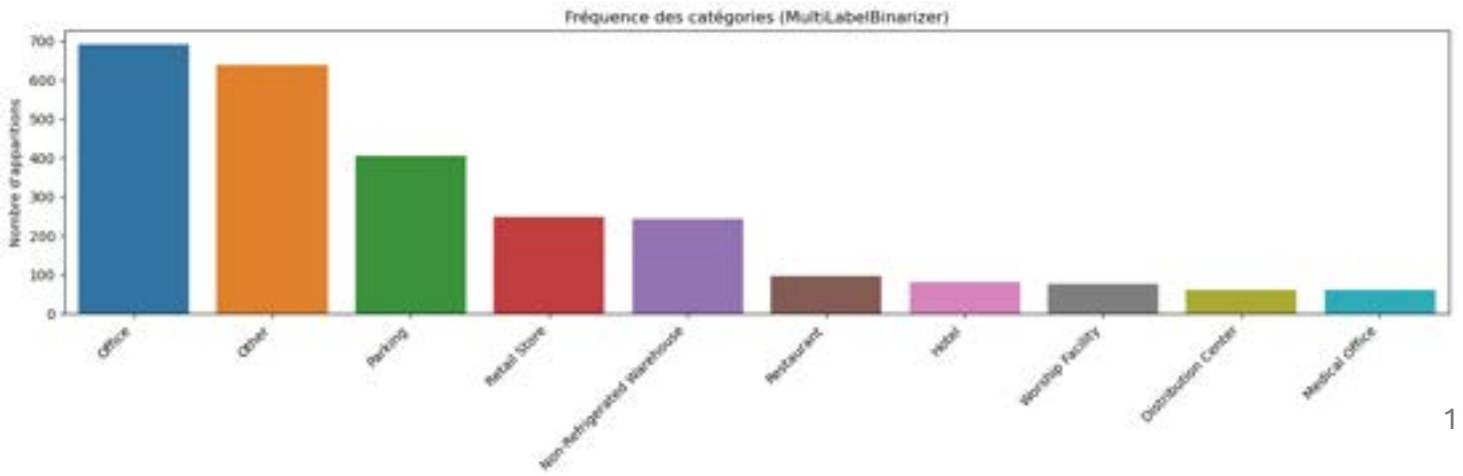
Encodage avec le MultiLabelBinarizer

	Distribution Center	Hotel	Medical Office	Non-Refrigerated Warehouse	Office	Other	Parking	Restaurant	Retail Store	Worship Facility
0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	1	0	
2	0	1	0	0	0	0	0	0	0	
3	0	1	0	0	0	0	0	0	0	
4	0	1	0	0	0	1	1	0	0	
...	...	...	...	...	...	...	...	...	...	
3338	0	0	0	1	0	0	0	0	0	
3339	0	0	0	0	1	0	0	0	0	
3340	0	0	0	0	0	1	0	0	0	
3347	0	0	0	0	0	1	1	0	0	
3356	0	0	0	0	1	1	1	0	0	

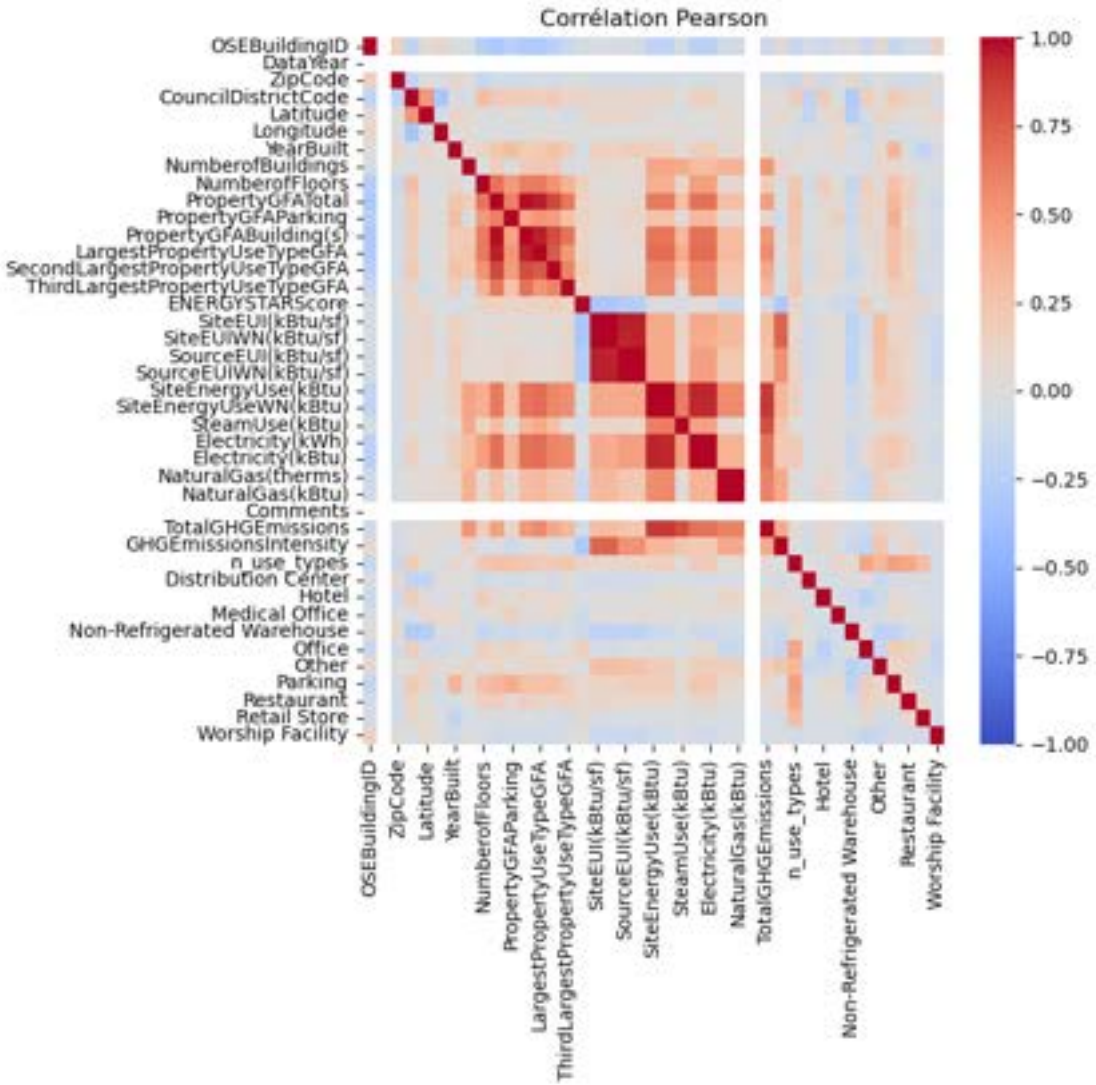
1460 rows × 10 columns

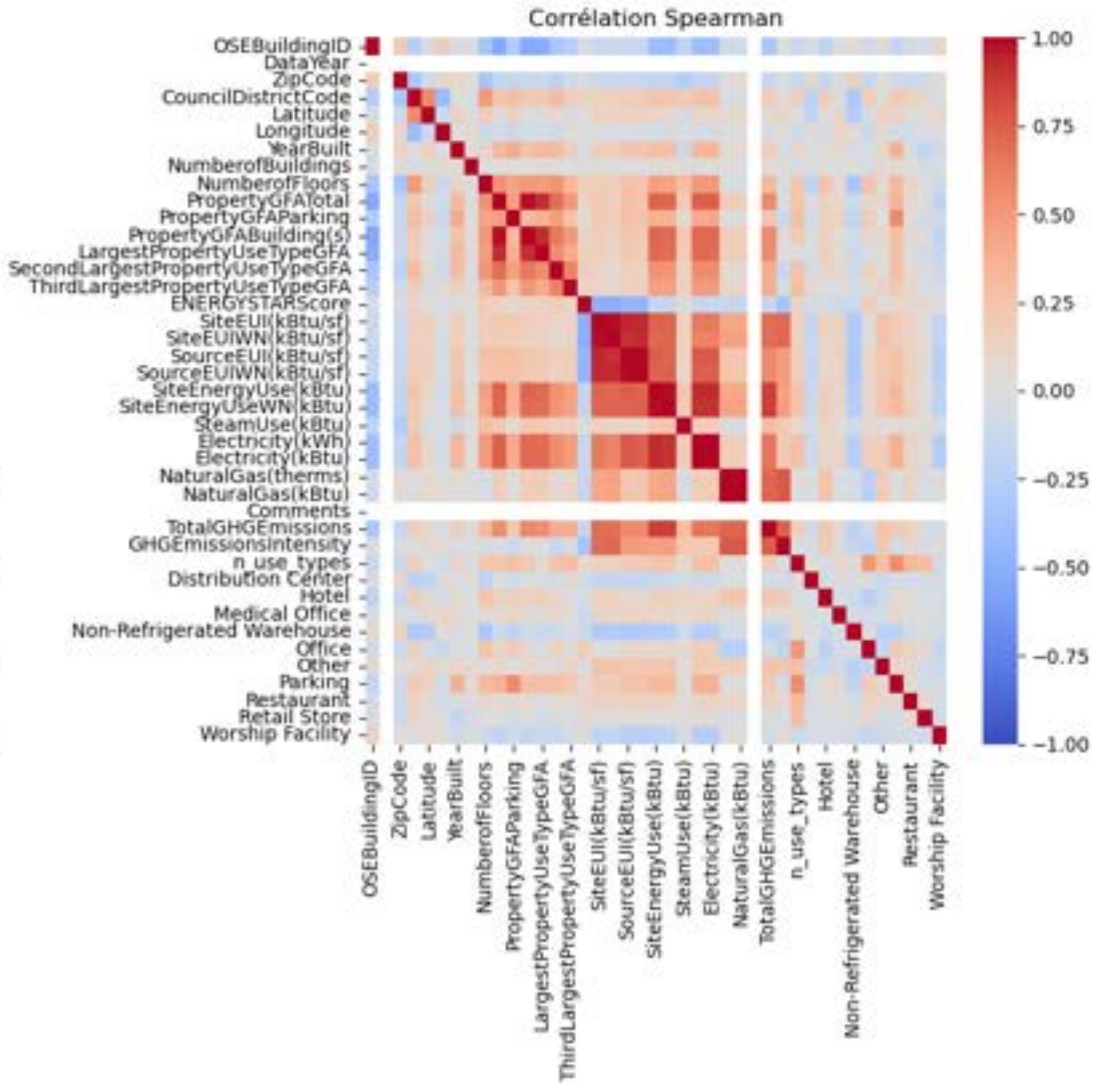
	count	percentage
Office	692	47.4
Other	639	43.8
Parking	406	27.8
Retail Store	250	17.1
Non-Refrigerated Warehouse	245	16.8
Restaurant	98	6.7
Hotel	81	5.5
Worship Facility	76	5.2
Distribution Center	61	4.2
Medical Office	61	4.2

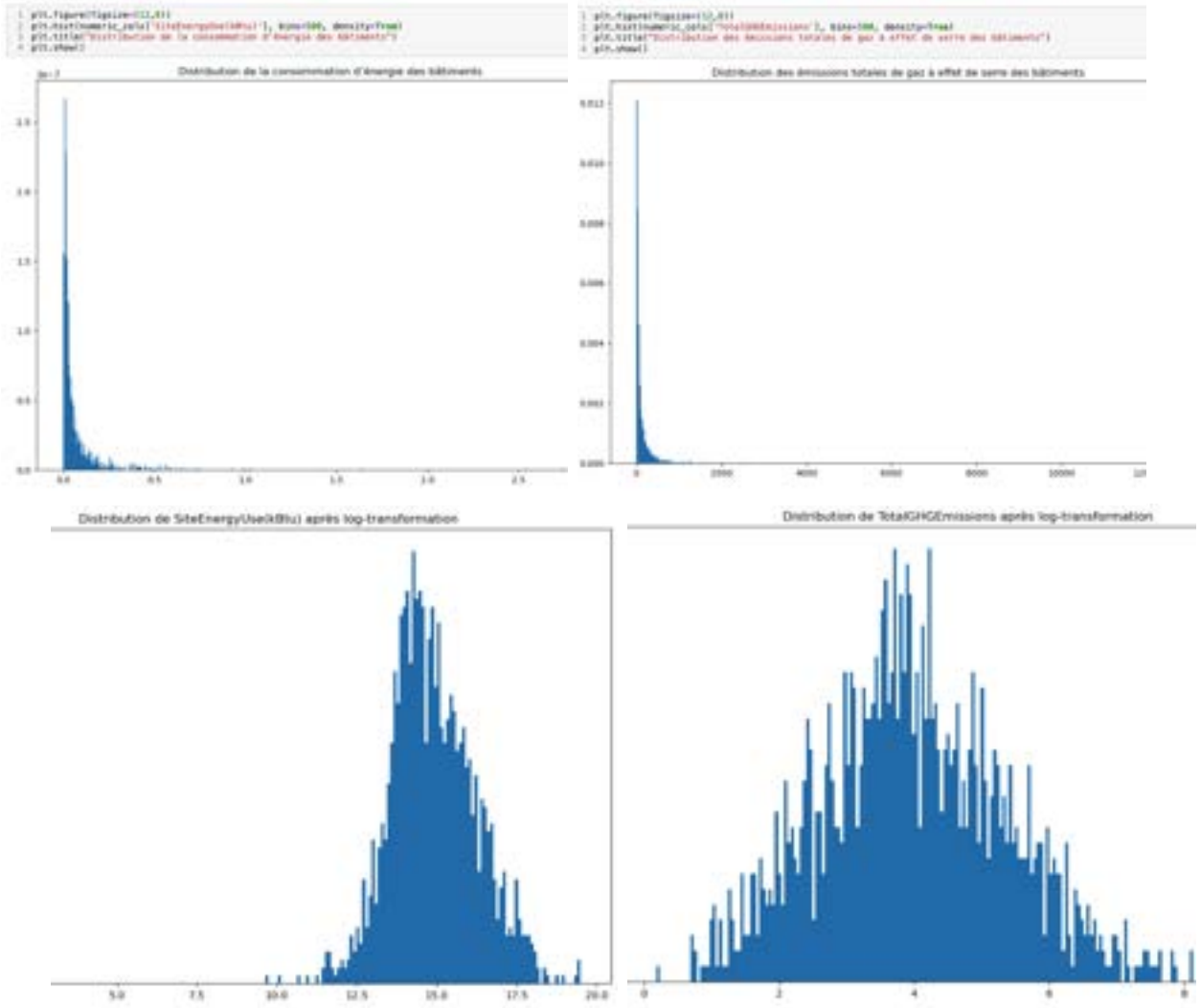
Dimensions du jeu de données avant =
(1460 lignes, 50 colonnes)
Dimensions après = (1460 lignes, 60 colonnes)



Matrices de corrélation







Distribution des variables cibles

- Asymétrie
- Transformation-log des variables cibles :
  - df['log_SiteEnergyUse(kBtu)'] = np.log1p(df['SiteEnergyUse(kBtu)']) # toutes les valeurs sont strictement positives
  - df['log_TotalGHGEmissions'] = np.log(df['TotalGHGEmissions']+2)

II. 2. c) Suppression des colonnes redondantes ou peu utiles

```
df.shape
```

```
(1460, 62)
```

```
df.drop(columns=['City', # n'affiche que la valeur 'Seattle' donc colonne non pertinente
                 'State', # idem
                 'DefaultData', # l'information apportée est déjà contenue dans la colonne ['ComplianceStatus']
                 'Comments' # ne comporte aucune donnée
                 ], inplace=True)
df.shape
```

```
(1460, 58)
```

```
df
```

Feature Engineering

But : Enrichir le jeu de données actuel avec de nouvelles features issues de celles existantes.

Les variables agrégées basées sur les émissions de CO_2 ou la consommation énergétique ont été exclues afin d'éviter toute fuite d'information, ces valeurs n'étant pas disponibles pour les bâtiments à prédire.

```python
# ---------------------------
# A. Géographie
# ---------------------------

## A.Distance au centre-ville
### Coordonnées du centre-ville de Seattle
Downtown_lat = 47.6062
Downtown_lon = -122.3321

def haversine_vec(lat1, lon1, lat2, lon2):
    R = 6371
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1)*np.cos(lat2)*np.sin(dlon/2)**2
    return 2 * R * np.arcsin(np.sqrt(a))

df["Dist_to_Downtown_km"] = haversine_vec(df['Latitude'],
                                          df['Longitude'],
                                          Downtown_lat,
                                          Downtown_lon)
```

```python
# ---------------------------
# B. Caractéristiques structurelles des bâtiments
# ---------------------------

## B.1. Âge du bâtiment
df['Building_Age'] = df['DataYear'] - df['YearBuilt']

## B.2. Superficie par étage
df['GFAPerFloors'] = safe_divide(df['PropertyGFATotal'], df['NumberofFloors'])

## B.3. % occupé par le parking
df['PctGFAParking'] = safe_divide(df['PropertyGFAParking'], df['PropertyGFATotal'])

## B.4. Source d'énergie
df['Has_Electricity'] = np.where(
    df['Electricity(kBtu)'].notna() & (df['Electricity(kBtu)'] > 0)
    ,1,0)

df['Has_NaturalGas'] = np.where(
    df['NaturalGas(kBtu)'].notna() & (df['NaturalGas(kBtu)'] > 0)
    ,1,0)

df['Has_Steam'] = np.where(
    df['SteamUse(kBtu)'].notna() & (df['SteamUse(kBtu)'] > 0)
    ,1,0)
```

```python
# ---------------------------
# C. Usage des bâtiments
# ---------------------------

## C.1. Bâtiments mono ou multi-usage
df['MonoMultiUsage'] = np.where(
    df['SecondLargestPropertyUseType'].isna() & df['ThirdLargestPropertyUseType'].isna() # les conditions : pas d'usage secondaire ou tertiaire autre que l'usage principal
    ,'mono' # valeur si les conditions sont remplies (ie bâtiment mono-usage)
    ,'multi' # valeur sinon (bâtiment multi-usage)
    )

## C.2. Ratios d'usages primaire et secondaire
df['PrimaryUseRatio'] = df['LargestPropertyUseTypeGFA'] / df['PropertyGFATotal'].replace(0, np.nan)
df['SecondaryUseRatio'] = df['SecondLargestPropertyUseTypeGFA'] / df['PropertyGFATotal'].replace(0, np.nan)
```

Suppression de colonnes

```python
# Suppression des colonnes non pertinentes pour la problématique (afin d'éviter le data leakage)

leakage_cols = [
    'SiteEnergyUseWN(kBtu)',
    'SiteEUI(kBtu/sf)',
    'SiteEUIWN(kBtu/sf)',
    'SourceEUI(kBtu/sf)',
    'SourceEUIWN(kBtu/sf)',
    'Electricity(kWh)',
    'Electricity(kBtu)',
    'NaturalGas(therms)',
    'NaturalGas(kBtu)',
    'SteamUse(kBtu)',
    'GHGEmissionsIntensity'
]

df_clean = df.drop(columns=leakage_cols)
```

Après ajout de colonnes (Feature engineering) et suppression de colonnes :

1460 lignes, 57 colonnes

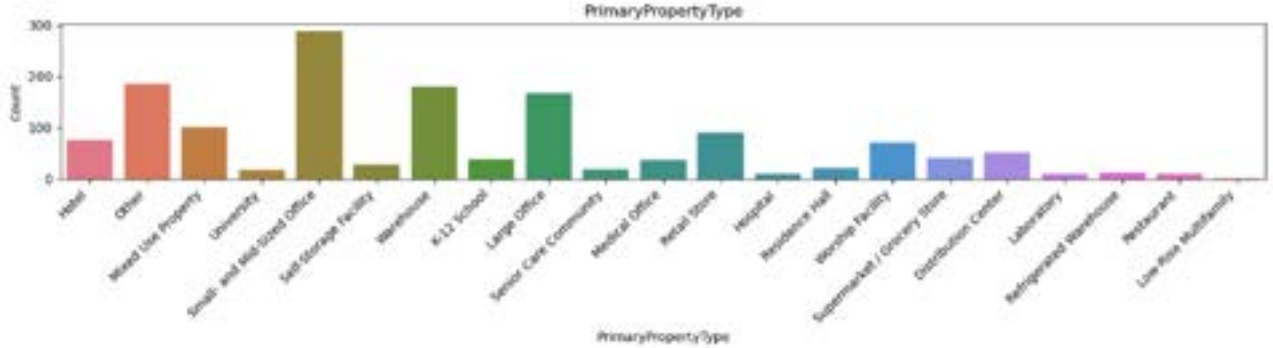
Dist_to_Downtown_km	Building_Age	GFAPerFloors	PctGFAParking	Has_Electricity	Has_NaturalGas	Has_Steam	PrimaryUseRatio	SecondaryUseRatio
1460.000000	1460.000000	1447.000000	1460.000000	1460.000000	1460.000000	1460.000000	1454.000000	796.000000
4.334082	55.156164	30991.457796	0.067473	0.995205	0.700000	0.067123	0.858865	0.236329
3.503303	32.966330	36872.634338	0.145862	0.069100	0.458415	0.250321	0.298228	0.168949
0.034028	1.000000	221.696970	0.000000	0.000000	0.000000	0.000000	0.186469	0.000000
1.309017	28.000000	12792.761905	0.000000	1.000000	0.000000	0.000000	0.675376	0.110472
3.351913	51.000000	21833.000000	0.000000	1.000000	1.000000	0.000000	0.940504	0.229458
6.524559	87.000000	35297.700000	0.000000	1.000000	1.000000	0.000000	1.000000	0.339705
14.242219	116.000000	530039.000000	0.895023	1.000000	1.000000	1.000000	6.426849	1.452054

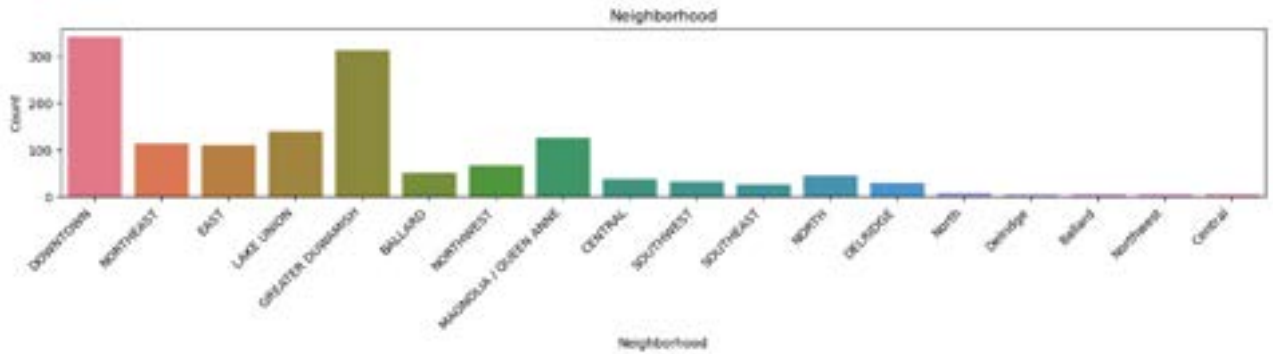
```
df_clean['Dist_to_dtown_bin'] = pd.qcut(
    df_clean['Dist_to_Downtown_km'],
    q=4,
    labels=['0–1.6 km', '1.6–3.7 km', '3.7–7.1 km', '7.1+ km']
)
```

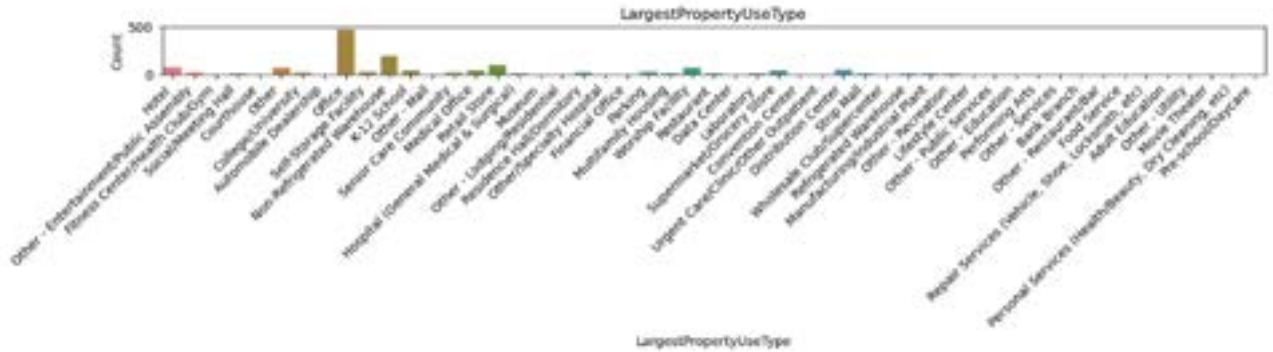
```
df_clean['Building_Age_bin'] = pd.cut(df_clean['Building_Age'],
                                      bins = [0, 20, 50, 80, 200],
                                      labels = ['<20', '20–50', '50–80', '80+'])
```

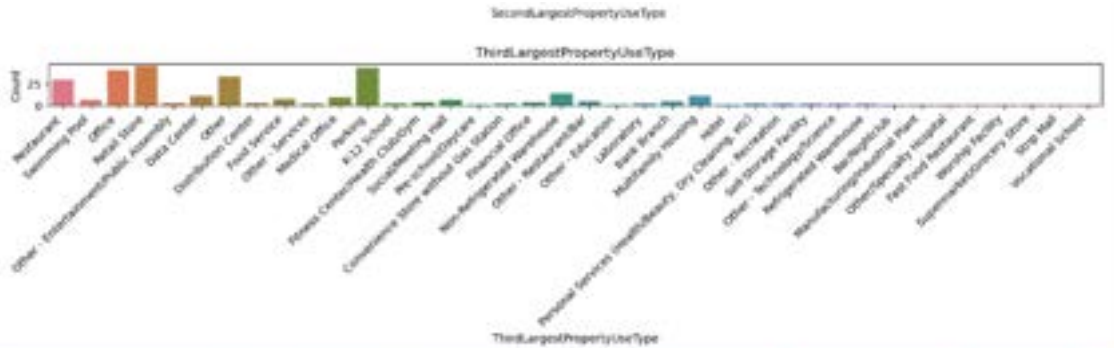
Dimensions:
1460 lignes, 59 colonnes

Simplification des colonnes catégorielles

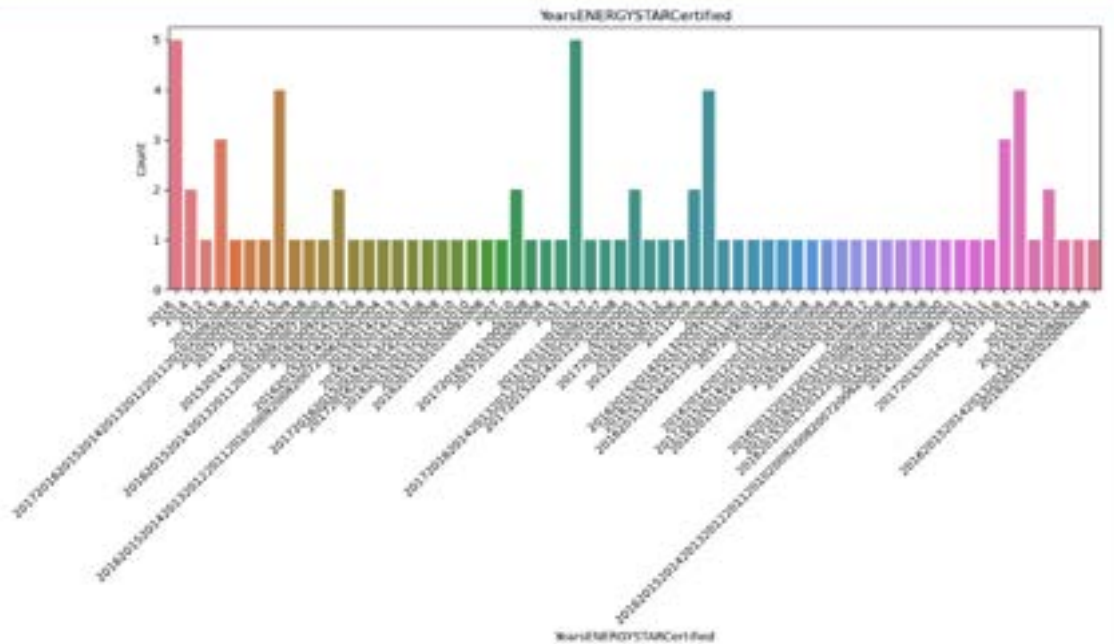






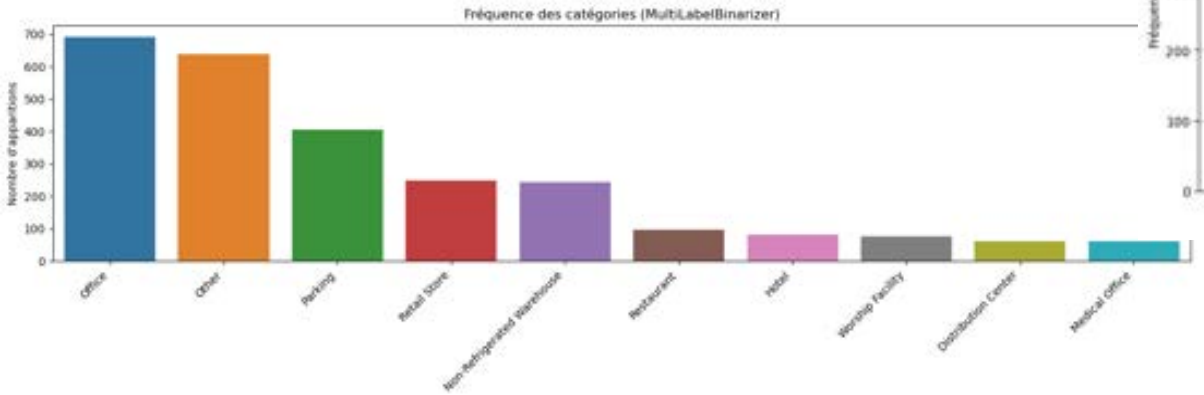


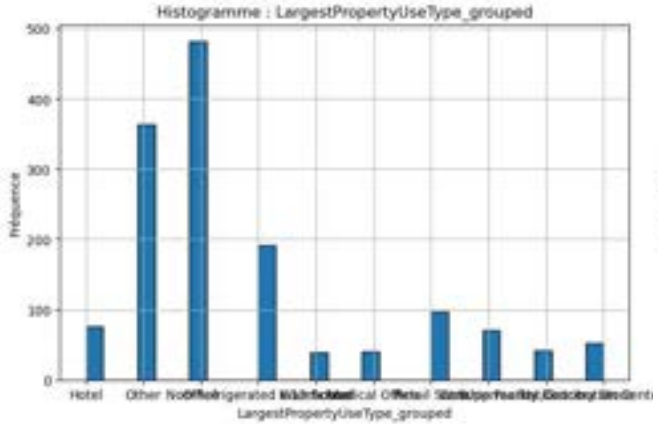

```
/var/folders/lw/0z6yvdmn5p1fcqf8r658c0sc0000gn/T/ipykernel_15876/111760156.py:20: UserWarning: Tight layout not app
lied. The bottom and top margins cannot be made large enough to accommodate all axes decorations.
  plt.tight_layout()
```

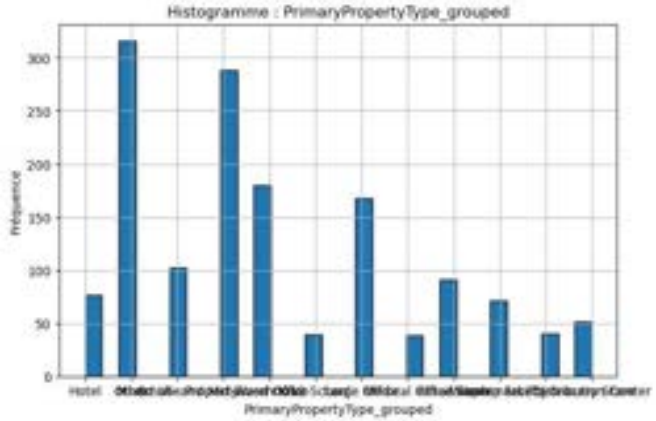


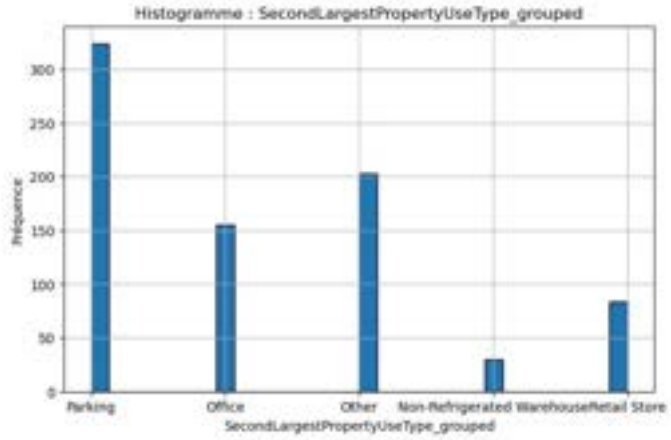
Simplification des colonnes catégorielles

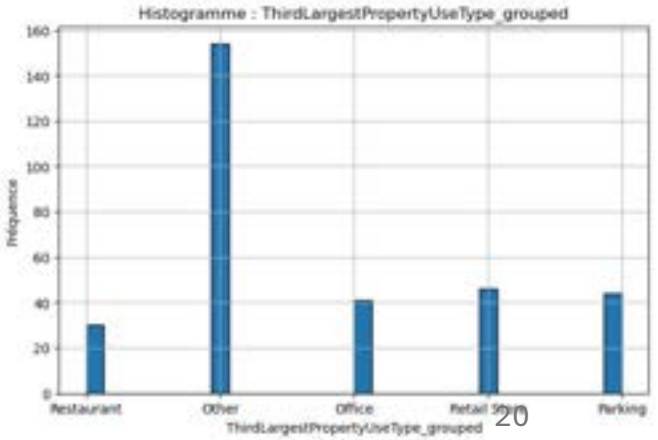
Après regroupement des catégories les moins fréquentes dans du 'Other' :











- Extraction des informations de la colonne `YearsENERGYSTARCertified`

La variable YearsENERGYSTARCertified étant un historique concaténé, elle a été transformée en indicateurs synthétiques décrivant la présence, la durée et la récence de la certification, afin de conserver l'information pertinente tout en évitant une explosion dimensionnelle.

```
1  df_clean['Has_ENERGYSTAR'] = df_clean['YearsENERGYSTARCertified'].notna().astype(int)
2
3  df_clean['ENERGYSTAR_Years_Count'] = df_clean['YearsENERGYSTARCertified'].apply(lambda x: len(str(x)) // 4 if pd.notna(x) else 0)
```

✓ 0.0s

```
1  print(df_clean['Has_ENERGYSTAR'].sum())
```

✓ 0.0s

Nettoyage

III. 5. Nettoyage

III. 5. a) des colonnes non utiles

```
# ========================
# Variables prédictives (X)
# ========================

X = df_clean.drop(columns=[
    # Identifiants et informations très spécifiques non généralisables
    'OSEBuildingID', 'TaxParcelIdentificationNumber',
    'PropertyName', 'Address',

    # Variables redondantes ou déjà agrégées
    'YearsENERGYSTARCertified',
    'Latitude', 'Longitude', 'DataYear', 'YearBuilt',
    'ListOfAllPropertyUseTypes', 'parsed', 'parsed_simple','parsed_simple_grouped',
    'LargestPropertyUseType','SecondLargestPropertyUseType','ThirdLargestPropertyUseType',

    # Variables causant du data leakage
    'SiteEnergyUse(kBtu)', 'TotalGHGEmissions',
    'log_SiteEnergyUse(kBtu)', 'log_TotalGHGEmissions',
    'ENERGYSTARScore', 'ComplianceStatus', 'Outlier'
]).copy()

# ====================
# Variables cibles (y)
# ====================

y_energy = df_clean['log_SiteEnergyUse(kBtu)'].fillna(0) # on impute 0 aux NaN
y_ghg = df_clean['log_TotalGHGEmissions'].fillna(0) # il faut éviter d'en avoir pour l'entraînement des modèles
```

1460 lignes et 65 colonnes
--> 1460 lignes et 42 colonnes

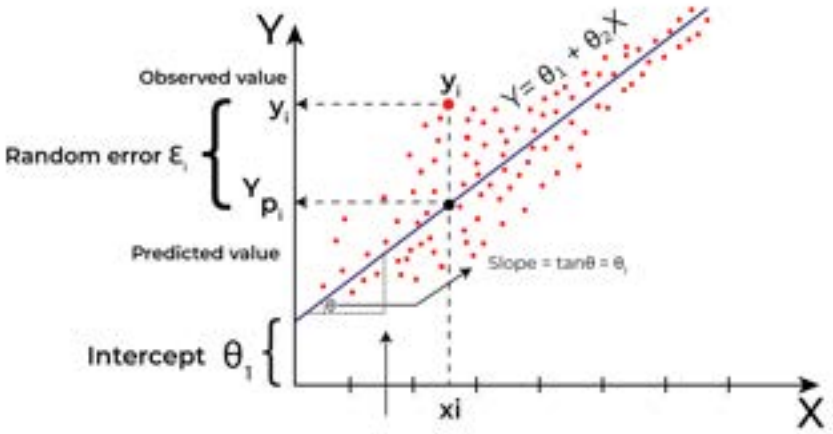
Modélisation

Modèles testés dans ce projet :

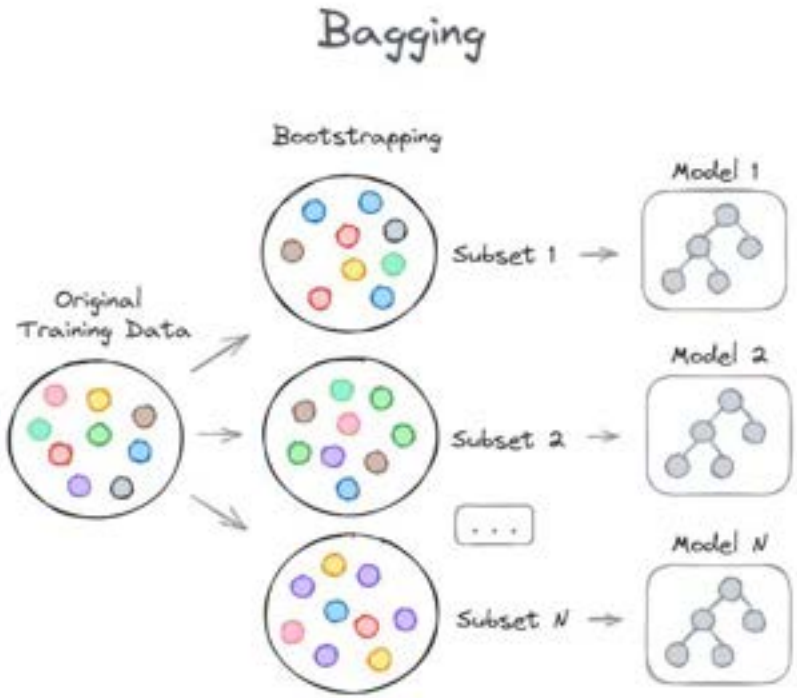
- Linear Regression
- Random forest (Bagging)
- Lightgbm (Boosting)

Explication des modèles utilisés

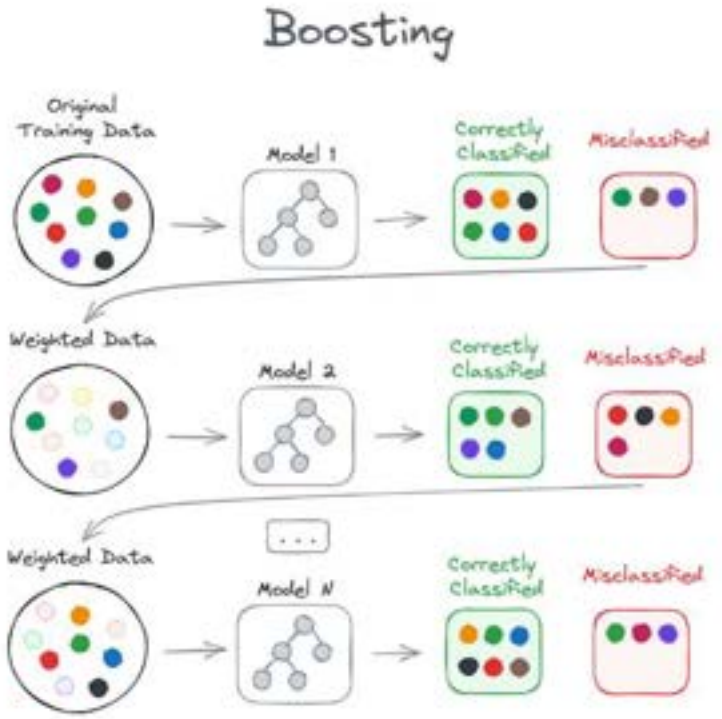
Linear regression



Ex : Random Forest



Ex : CatBoost, AdaBoost, XGBoost, LightGBM…



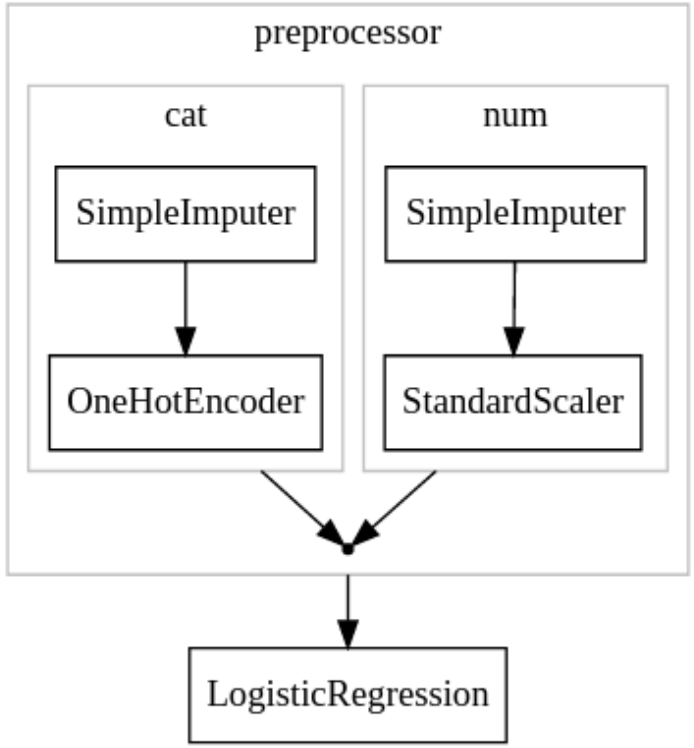
Préparation à la modélisation

- Attribution manuelle des types de colonnes (catégorielles, numériques continues/discrètes)
- Vérification qu'il n'y a aucun NaN dans les variables cibles y_ghg et y_energy
- # 1. Création de pipeline (sélection de modèle & preprocessing)
- # 2. Train / Test split
- # 3. Cross-validation sur le train

```
# Attribution manuelle des types de colonnes
categorical_features = [
    'BuildingType',
    'ZipCode',
    'CouncilDistrictCode',
    'Neighborhood',
    'PrimaryPropertyType_grouped',
    'LargestPropertyUseType_grouped',
    'SecondLargestPropertyUseType_grouped',
    'ThirdLargestPropertyUseType_grouped',
    'MonoMultiUsage',
    'Building_Age_bin',
    'Dist_to_dtown_bin'
]

num_continuous_features  = [
    'LargestPropertyUseTypeGFA', # surface = 0 si l'usage de la propriété n'est pas défi
    'SecondLargestPropertyUseTypeGFA',
    'ThirdLargestPropertyUseTypeGFA',
    'PropertyGFATotal',
    'PropertyGFAParking',
    'PropertyGFABuilding(s)',
    'Dist_to_Downtown_km',
    'GFAPerFloors',
    'PctGFAParking',
    'Building_Age',
    'PrimaryUseRatio',
    'SecondaryUseRatio'
]

num_discrete_features = [
    'NumberofBuildings',
    'NumberofFloors',
    'n_use_types',
    'Distribution Center',
    'Hotel',
    'Medical Office',
    'Office',
    'Non-Refrigerated Warehouse',
    'Other',
    'Parking',
    'Restaurant',
    'Retail Store',
    'Worship Facility',
    'ENERGYSTAR_Years_Count', # 0 année de labellisation si aucune
    'Has_Electricity',
    'Has_NaturalGas',
    'Has_Steam',
    'Has_ENERGYSTAR'
]
```

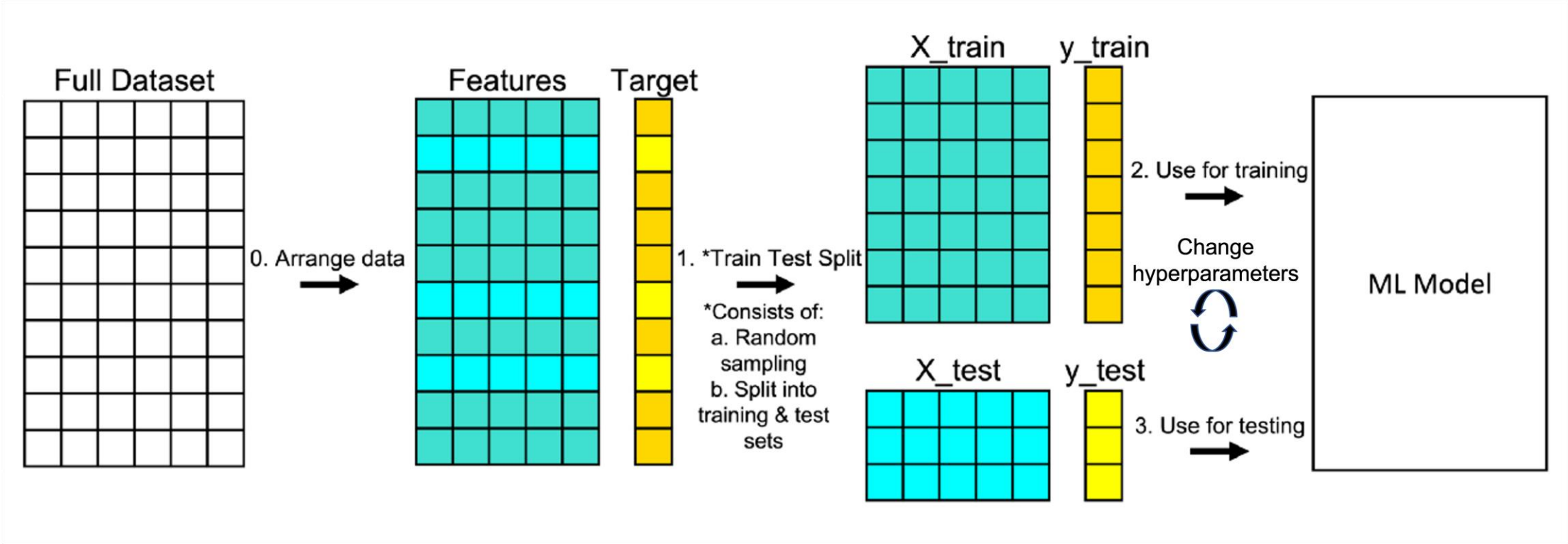
1. Création de pipeline (sélection de modèle & preprocessing)

Exemple :




```python
def make_pipeline(model_name):

    if model_name == "Linear Regression":
        model = LinearRegression()
        preprocessor = ColumnTransformer(
            transformers=[
                ('cat',
                 Pipeline([
                     ('imputer', SimpleImputer(strategy='constant', fill_value='Not_known')),
                     ('encoder', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
                 ]),
                 categorical_features),

                ('num',
                 Pipeline([
                     ('imputer', SimpleImputer(strategy='constant', fill_value=0)),
                     ('scaler', StandardScaler())
                 ]),
                 numeric_features)
            ]
        )

    elif model_name == "Random Forest":
        model = RandomForestRegressor(random_state=42)
        preprocessor = ColumnTransformer(
            transformers=[
                ('cat',
                 Pipeline([
                     ('imputer', SimpleImputer(strategy='constant', fill_value='Not_known')),
                     ('encoder', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
                 ]),
                 categorical_features),

                ('num', SimpleImputer(strategy='constant', fill_value=0), numeric_features)
            ]
        )

    elif model_name == "LightGBM":
        model = lgb.LGBMRegressor(random_state=42)
        preprocessor = ColumnTransformer(
            transformers=[
                ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
                ('num', 'passthrough', numeric_features)
            ]
        )

    else:
        raise ValueError("Modèle inconnu")
```

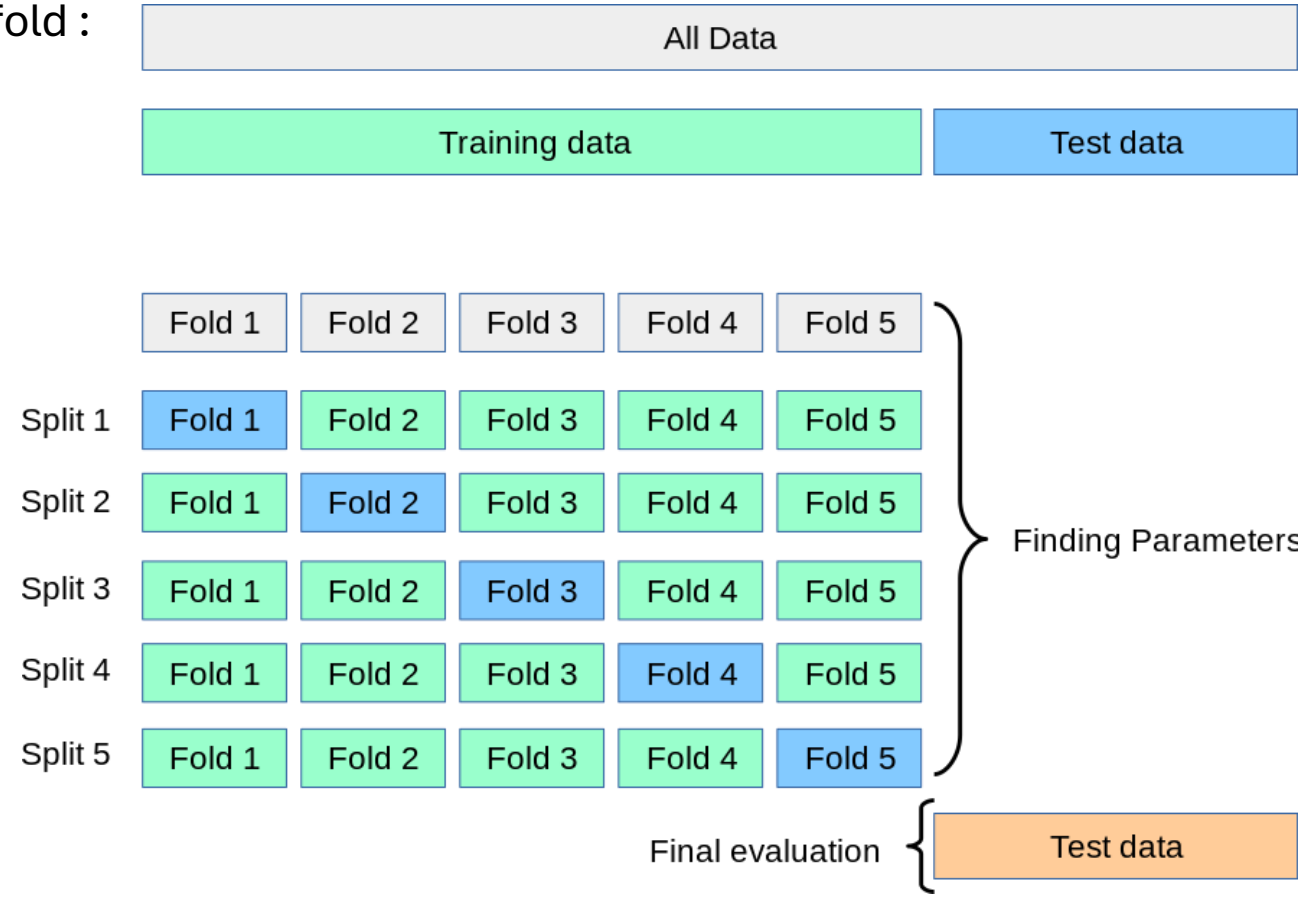
2. Train-test split



3. Cross-validation

Cross validation

Kfold :



Plusieurs types de Cross-validation :

- Kfold <--- Celui qu'on utilise
- Stratified
- Temporel
- etc.

4. Entrainement final & évaluation sur le test

5. Boucle d'expérimentation & Résultats

```python
def test_pipeline(pipe, X_train, X_test, y_train, y_test):

    pipe.fit(X_train, y_train)    # Transformation (pipe) + entrainement sur le 'train'
    y_pred = pipe.predict(X_test) # Transformation (pipe) + Prédiction (predict)

    return {
        "r2": r2_score(y_test, y_pred),
        "rmse": np.sqrt(mean_squared_error(y_test, y_pred)),
        "mae": mean_absolute_error(y_test, y_pred)
    }
```

```python
results = []

targets = {
    "ghg": y_ghg,
    "energy": y_energy
}

models = ["Linear Regression", "Random Forest", "LightGBM"]

metrics = {
    "r2": "r2", # r2 est à maximiser (plus on est proche de 1, meilleure est la performa
    "rmse": "neg_root_mean_squared_error", # RMSE est à minimiser, donc le RMSE négatif
    "mae": "neg_mean_absolute_error" # pareil que pour RMSE, MAE négatif est à maximiser
}

for target_name, y in targets.items():

    X_train, X_test, y_train, y_test = split_data(X, y) # Train/test split pour chaque c

    for model_name in models:

        pipe = make_pipeline(model_name)

        # Cross-validation uniquement sur le 'train'
        cv_scores = evaluate_pipeline(
            pipe,
            X_train,
            y_train,

            metrics=list(metrics.values())
        )

        # Entrainement final + évaluation sur le test
        test_scores = test_pipeline(pipe, X_train, X_test, y_train, y_test)

        for metric_name, sk_metric in metrics.items():
            results.append({
                "target": target_name,
                "model": model_name,
                "metric": metric_name,
                "cv_mean": -cv_scores[f"test_{sk_metric}"].mean() # on ajoute un '-' pou
                           if "neg" in sk_metric                  # ... dans le cas des
                           else cv_scores[f"test_{sk_metric}"].mean(),
                "test_score": test_scores[metric_name]
            })
```

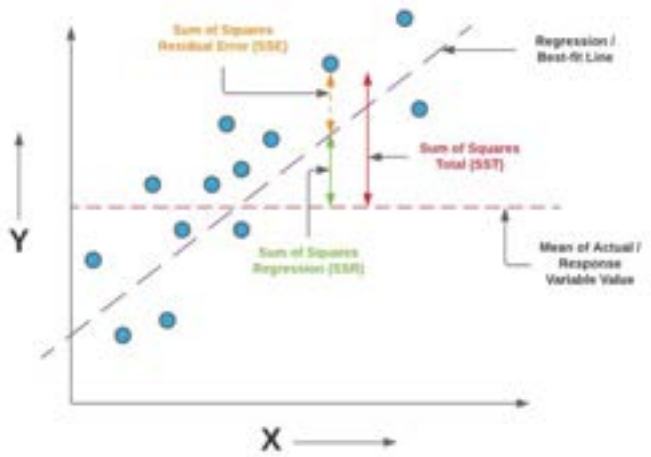
Résultats

	target	model	metric	cv_mean	test_score
14	energy	Random Forest	mae	0.567923	0.673914
17	energy	LightGBM	mae	0.614730	0.710517
11	energy	Linear Regression	mae	0.724582	0.771033
9	energy	Linear Regression	r2	0.386938	0.421793
15	energy	LightGBM	r2	0.515830	0.352358
12	energy	Random Forest	r2	0.553300	0.577418
13	energy	Random Forest	rmse	0.977702	1.158257
16	energy	LightGBM	rmse	1.021111	1.433895
10	energy	Linear Regression	rmse	1.100206	1.354851
5	ghg	Random Forest	mae	0.608366	0.639163
8	ghg	LightGBM	mae	0.625504	0.648903
2	ghg	Linear Regression	mae	0.724340	0.740163
0	ghg	Linear Regression	r2	0.534707	0.546206
6	ghg	LightGBM	r2	0.645173	0.666107
3	ghg	Random Forest	r2	0.659021	0.678984
4	ghg	Random Forest	rmse	0.813365	0.825704
7	ghg	LightGBM	rmse	0.829325	0.842102
1	ghg	Linear Regression	rmse	0.950434	0.981726

Métrique d'évaluation

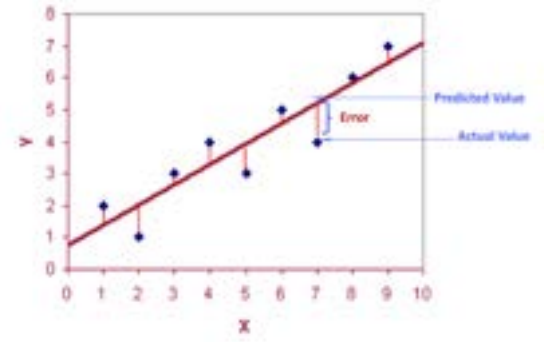
Métriques d’évaluation

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$



- R^2 (coefficient de détermination)
  Mesure la part de variance de la cible expliquée par le modèle. $R^2 = 1$ → parfait, $R^2 = 0$ → équivalent à une moyenne, $R^2 < 0$ → pire qu’un modèle naïf.
  ✅ Intuitif et comparable entre modèles • ⚠️ Sensible aux outliers, ne renseigne pas sur l’erreur absolue.

- RMSE (Root Mean Squared Error) (La racine de l'erreur quadratique moyenne)
  Racine de l'Erreur moyenne quadratique (même unité que la cible). Pénalise fortement les grandes erreurs.
  ✅ Utile quand les grosses erreurs sont critiques • ⚠️ Très sensible aux outliers, moins intuitif.



- MAE (Mean Absolute Error)
  Erreur moyenne absolue (même unité que la cible) : “en moyenne, le modèle se trompe de X unités”.
  ✅ Facile à interpréter, robuste aux outliers • ⚠️ Pénalise moins les grosses erreurs.

➡️ *En cross-validation, RMSE et MAE sont utilisés sous forme négative (neg_)* pour permettre la maximisation des scores.

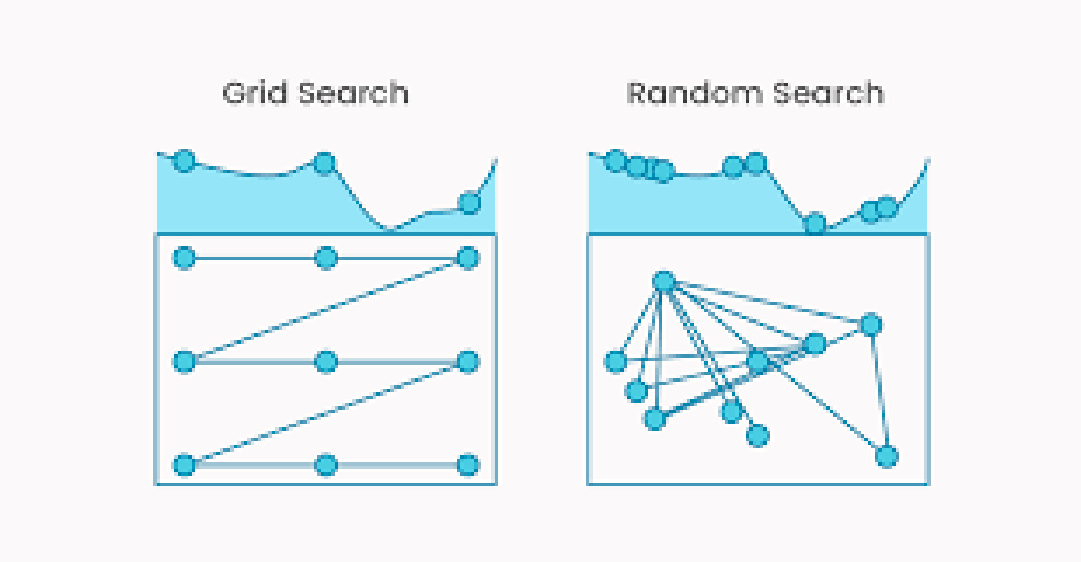
Résultats et interprétations

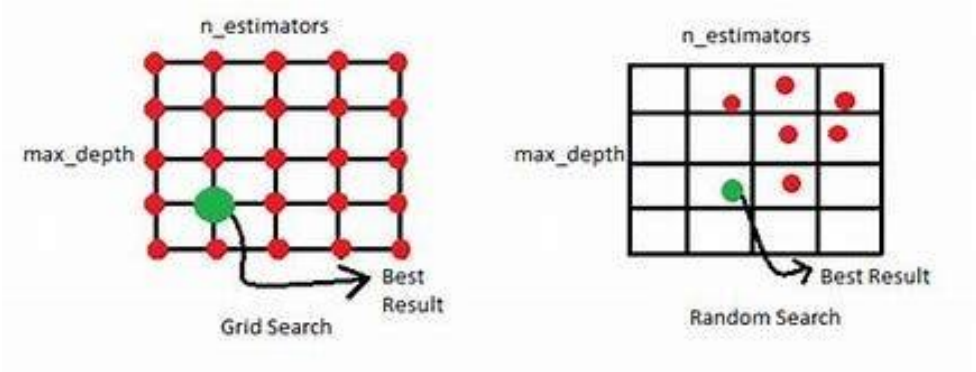
	target	model	metric	cv_mean	test_score
14	energy	Random Forest	mae	0.567923	0.673914
17	energy	LightGBM	mae	0.614730	0.710517
11	energy	Linear Regression	mae	0.724582	0.771033
9	energy	Linear Regression	r2	0.386938	0.421793
15	energy	LightGBM	r2	0.515830	0.352358
12	energy	Random Forest	r2	0.553300	0.577418
13	energy	Random Forest	rmse	0.977702	1.158257
16	energy	LightGBM	rmse	1.021111	1.433895
10	energy	Linear Regression	rmse	1.100206	1.354851
5	ghg	Random Forest	mae	0.608366	0.639163
8	ghg	LightGBM	mae	0.625504	0.648903
2	ghg	Linear Regression	mae	0.724340	0.740163
0	ghg	Linear Regression	r2	0.534707	0.546206
6	ghg	LightGBM	r2	0.645173	0.666107
3	ghg	Random Forest	r2	0.659021	0.678984
4	ghg	Random Forest	rmse	0.813365	0.825704
7	ghg	LightGBM	rmse	0.829325	0.842102
1	ghg	Linear Regression	rmse	0.950434	0.981726

	target	metric	model	test_score
3	ghg	r2	Random Forest	0.678984
4	ghg	rmse	Random Forest	0.825704
5	ghg	mae	Random Forest	0.639163
12	energy	r2	Random Forest	0.577418
13	energy	rmse	Random Forest	1.158257
14	energy	mae	Random Forest	0.673914

Recherche des meilleurs hyperparamètres

Recherche d'hyperparamètres





- Recherche d'hyperparamètres avec GridSearchCV
- RandomSearchCV

Exemples : LightGBM

```
# Petite grille
lgb_param_grid = {
    "model__n_estimators": [100, 300],
    "model__learning_rate": [0.05, 0.1],
    "model__max_depth": [-1, 10]
}
```

Random Forest

```
# 3. GridSearch (sur le train uniquement)
param_grid_rf = {
    "model__n_estimators": [200, 500],
    "model__max_depth": [None, 10],
    "model__min_samples_leaf": [1, 5]
}

grid_rf = GridSearchCV(
    pipe_rf,
    param_grid=param_grid_rf,
    cv=3,
    scoring="r2",
    n_jobs=-1
)

grid_rf.fit(X_train, y_train)
```

Meilleurs hyperparamètres avec GridSearchCV sur le Random Forest

```python
# 3. GridSearch (sur le train uniquement)
param_grid_rf = {
    "model__n_estimators": [200, 500],
    "model__max_depth": [None, 10],
    "model__min_samples_leaf": [1, 5]
}

grid_rf = GridSearchCV(
    pipe_rf,
    param_grid=param_grid_rf,
    cv=3,
    scoring="r2",
    n_jobs=-1
)

grid_rf.fit(X_train, y_train)
```

Meilleurs hyperparamètres Random Forest pour y_ghg:

{'model__max_depth': 10,
'model__min_samples_leaf': 1,
'model__n_estimators': 500}

Meilleur R^2 CV : 0.6553668060520381

R2 test : 0.6803859927051854
RMSE test : 0.8238986220318549
MAE test : 0.6356618517409269

Meilleurs hyperparamètres Random Forest pour y_energy :

{'model__max_depth': 10,
'model__min_samples_leaf': 5,
'model__n_estimators': 500}

Meilleur R² CV : 0.55728061533576

R2 test : 0.40277204485321527
RMSE test : 1.3769549094256728
MAE test : 0.6810861903284482

target	metric	model	test_score
ghg	r2	Random Forest	0.678984
ghg	rmse	Random Forest	0.825704
ghg	mae	Random Forest	0.639163

energy	r2	Random Forest	0.577418
energy	rmse	Random Forest	1.158257
energy	mae	Random Forest	0.673914

CONCLUSION

Identification des features les plus impactantes sur la performance du modèle

	feature	importance
137	PropertyGFATotal	0.607135
134	LargestPropertyUseTypeGFA	0.050754
139	PropertyGFABuilding(s)	0.043233
141	GFAPerFloors	0.040413
140	Dist_to_Downtown_km	0.033897
81	PrimaryPropertyType_grouped_Warehouse	0.022666
131	Has_NaturalGas	0.021492
143	Building_Age	0.020705
92	LargestPropertyUseType_grouped_Supermarket/Gro...	0.018901
80	PrimaryPropertyType_grouped_Supermarket / Groc...	0.018597